

Communications Blockset

For Use with Simulink[®]

Computation
└─

Visualization
└─

Programming
└─



User's Guide

Version 2

How to Contact The MathWorks:



508-647-7000

Phone



508-647-7001

Fax



The MathWorks, Inc.
3 Apple Hill Drive
Natick, MA 01760-2098

Mail



<http://www.mathworks.com>
<ftp.mathworks.com>
<comp.soft-sys.matlab>

Web
Anonymous FTP server
Newsgroup



support@mathworks.com
suggest@mathworks.com
bugs@mathworks.com
doc@mathworks.com
subscribe@mathworks.com
service@mathworks.com
info@mathworks.com

Technical support
Product enhancement suggestions
Bug reports
Documentation error reports
Subscribing user registration
Order status, license renewals, passcodes
Sales, pricing, and general information

Communications Blockset User's Guide

© COPYRIGHT 2000 by The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by or for the federal government of the United States. By accepting delivery of the Program, the government hereby agrees that this software qualifies as "commercial" computer software within the meaning of FAR Part 12.212, DFARS Part 227.7202-1, DFARS Part 227.7202-3, DFARS Part 252.227-7013, and DFARS Part 252.227-7014. The terms and conditions of The MathWorks, Inc. Software License Agreement shall pertain to the government's use and disclosure of the Program and Documentation, and shall supersede any conflicting contractual terms or conditions. If this license fails to meet the government's minimum needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to MathWorks.

MATLAB, Simulink, Stateflow, Handle Graphics, and Real-Time Workshop are registered trademarks, and Target Language Compiler is a trademark of The MathWorks, Inc.

Other product or brand names are trademarks or registered trademarks of their respective holders.

Printing History: September 2000

New for Version 2 (Release 12)

Preface

What Is the Communications Blockset?	xii
Related Products	xiii
Using This Guide	xv
Expected Background	xv
Organization of the Document	xvi
Configuration Information	xvii
Using the Blockset	xvii
Technical Conventions	xviii
Scalars, Vectors, and Matrices	xviii
Frame-Based and Sample-Based Signals	xviii
Typographical Conventions	xx

Getting Started with the Communications Blockset

1

The Example Model	1-3
Overview of the Simulation	1-3
Building the Model	1-5
Exploring the Model	1-9
Components of the Example	1-11
Creating Random Binary Data	1-11
Encoding Using a Convolutional Code	1-12
Unbuffering to Convert Vectors to Scalars	1-14
Modulating the Encoded Messages	1-16

Transmitting Along a Noisy Channel	1-16
Mapping the Received Data	1-18
Buffering to Convert Scalars to Vectors	1-20
Decoding the Convolutional Code	1-21
Computing the Error Rate	1-24
Displaying the Error Rate	1-26
Other Blocks	1-26
Learning More About the Example	1-27
Modifying the Model	1-27
For Further Study	1-28

Using the Communications Blockset

2

Signal Support	2-3
Processing Vectors and Matrices	2-3
Processing Frame-Based and Sample-Based Signals	2-4
Communications Sources	2-6
Source Features of the Blockset	2-6
Random or Pseudorandom Signals	2-6
Nonrandom Signals	2-11
Communications Sinks	2-12
Sink Features of the Blockset	2-12
Writing to a File	2-12
Error Statistics	2-12
Eye Diagrams	2-13
Scatter Diagrams	2-13
Example: Using Eye and Scatter Diagrams	2-14
Source Coding	2-16
Source Coding Features of the Blockset	2-16
Representing Quantization Parameters	2-17
Quantizing a Signal	2-17
Implementing Differential Pulse Code Modulation	2-21

Companding a Signal	2-24
Selected Bibliography for Source Coding	2-26
Block Coding	2-27
Organization of This Section	2-27
Accessing Block Coding Blocks	2-27
Block Coding Features of the Blockset	2-28
Communications Toolbox Support Functions	2-29
Channel Coding Terminology	2-29
Data Formats for Block Coding	2-29
Using Block Encoders and Decoders Within a Model	2-32
Examples of Block Coding	2-32
Notes on Specific Block Coding Techniques	2-35
Selected Bibliography for Block Coding	2-39
Convolutional Coding	2-40
Organization of This Section	2-40
Accessing Convolutional Coding Blocks	2-40
Convolutional Coding Features of the Blockset	2-40
Parameters for Convolutional Coding	2-41
Examples of Convolutional Coding	2-42
Selected Bibliography for Convolutional Coding	2-45
Interleaving	2-46
Interleaving Features of the Blockset	2-46
Block Interleavers	2-46
Convolutional Interleavers	2-47
Selected Bibliography for Interleaving	2-51
Analog Modulation	2-53
Accessing Analog Modulation Blocks	2-53
Analog Modulation Features of the Blockset	2-53
Baseband Modulated Signals Defined	2-54
Representing Signals for Analog Modulation	2-55
Timing Issues in Analog Modulation	2-55
Filter Design Issues	2-59
Digital Modulation	2-64
Accessing Digital Modulation Blocks	2-64

Digital Modulation Features of the Blockset	2-65
Representing Signals for Digital Modulation	2-68
Delays in Digital Modulation	2-69
Upsampled Signals and Rate Changes	2-72
Examples of Digital Modulation	2-75
Selected Bibliography for Digital Modulation	2-83
Channels	2-84
Channel Features of the Blockset	2-84
AWGN Channel	2-84
Fading Channels	2-85
Binary Symmetric Channel	2-88
Selected Bibliography for Channels	2-89
Synchronization	2-90
Synchronization Features of the Blockset	2-90
Overview of PLL Simulation	2-91
Implementing an Analog Baseband PLL	2-91
Implementing a Digital PLL	2-92
Selected Bibliography for Synchronization	2-92

Function Reference

3

Alphabetical List of Functions	3-2
comm_links	3-3
commlib	3-4

Block Reference

4

Communications Sources	4-3
Communications Sinks	4-5

Source Coding	4-7
Channel Coding	4-9
Block Coding	4-9
Convolutional Coding	4-11
Interleaving	4-13
Block Interleaving	4-13
Convolutional Interleaving	4-15
Modulation	4-18
Digital Baseband Modulation	4-18
Analog Baseband Modulation	4-24
Digital Passband Modulation	4-26
Analog Passband Modulation	4-31
Channels	4-34
Synchronization	4-36
Basic Communications Functions	4-37
Integrators	4-37
Sequence Operations	4-38
Utility Functions	4-41
Alphabetical List of Blocks	4-42
A-Law Compressor	4-47
A-Law Expander	4-49
Algebraic Deinterleaver	4-51
Algebraic Interleaver	4-53
APP Decoder	4-56
AWGN Channel	4-60
Baseband PLL	4-64
BCH Decoder	4-66
BCH Encoder	4-68
Bernoulli Random Binary Generator	4-70
Binary Cyclic Decoder	4-73
Binary Cyclic Encoder	4-75

Binary-Input RS Encoder	4-77
Binary Linear Decoder	4-79
Binary Linear Encoder	4-81
Binary-Output RS Decoder	4-82
Binary Symmetric Channel	4-84
Binary Vector Noise Generator	4-86
Bit to Integer Converter	4-89
BPSK Demodulator Baseband	4-90
BPSK Modulator Baseband	4-92
Charge Pump PLL	4-94
Complex Phase Difference	4-97
Complex Phase Shift	4-98
Continuous-Time Eye and Scatter Diagrams	4-99
Convolutional Deinterleaver	4-103
Convolutional Encoder	4-105
Convolutional Interleaver	4-107
CPFSK Demodulator Baseband	4-109
CPFSK Demodulator Passband	4-112
CPFSK Modulator Baseband	4-115
CPFSK Modulator Passband	4-118
CPM Demodulator Baseband	4-121
CPM Demodulator Passband	4-125
CPM Modulator Baseband	4-130
CPM Modulator Passband	4-134
Data Mapper	4-139
DBPSK Demodulator Baseband	4-142
DBPSK Modulator Baseband	4-144
Deinterlacer	4-146
Derepeat	4-147
Descrambler	4-150
Differential Decoder	4-152
Differential Encoder	4-153
Discrete Modulo Integrator	4-154
Discrete-Time Eye and Scatter Diagrams	4-156
Discrete-Time VCO	4-159
DPCM Decoder	4-161
DPCM Encoder	4-163
DQPSK Demodulator Baseband	4-165
DQPSK Modulator Baseband	4-167
DSB AM Demodulator Baseband	4-171

DSB AM Demodulator Passband	4-173
DSB AM Modulator Baseband	4-175
DSB AM Modulator Passband	4-176
DSBSC AM Demodulator Baseband	4-178
DSBSC AM Demodulator Passband	4-180
DSBSC AM Modulator Baseband	4-182
DSBSC AM Modulator Passband	4-183
Enabled Quantizer Encode	4-185
Error Rate Calculation	4-187
FM Demodulator Baseband	4-194
FM Demodulator Passband	4-196
FM Modulator Baseband	4-198
FM Modulator Passband	4-200
Gaussian Noise Generator	4-202
General Block Deinterleaver	4-205
General Block Interleaver	4-207
General Multiplexed Deinterleaver	4-208
General Multiplexed Interleaver	4-210
General QAM Demodulator Baseband	4-212
General QAM Demodulator Passband	4-214
General QAM Modulator Baseband	4-217
General QAM Modulator Passband	4-219
GMSK Demodulator Baseband	4-222
GMSK Demodulator Passband	4-225
GMSK Modulator Baseband	4-228
GMSK Modulator Passband	4-231
Hamming Decoder	4-234
Hamming Encoder	4-236
Helical Deinterleaver	4-238
Helical Interleaver	4-241
Insert Zero	4-244
Integer-Input RS Encoder	4-246
Integer-Output RS Decoder	4-248
Integer to Bit Converter	4-250
Integrate and Dump	4-251
Interlacer	4-253
Linearized Baseband PLL	4-254
Matrix Deinterleaver	4-256
Matrix Helical Scan Deinterleaver	4-257
Matrix Helical Scan Interleaver	4-259

Matrix Interleaver	4-262
M-DPSK Demodulator Baseband	4-264
M-DPSK Demodulator Passband	4-267
M-DPSK Modulator Baseband	4-270
M-DPSK Modulator Passband	4-274
M-FSK Demodulator Baseband	4-277
M-FSK Demodulator Passband	4-280
M-FSK Modulator Baseband	4-283
M-FSK Modulator Passband	4-286
Modulo Integrator	4-289
M-PAM Demodulator Baseband	4-290
M-PAM Demodulator Passband	4-293
M-PAM Modulator Baseband	4-297
M-PAM Modulator Passband	4-301
M-PSK Demodulator Baseband	4-305
M-PSK Demodulator Passband	4-308
M-PSK Modulator Baseband	4-311
M-PSK Modulator Passband	4-316
MSK Demodulator Baseband	4-319
MSK Demodulator Passband	4-321
MSK Modulator Baseband	4-324
MSK Modulator Passband	4-326
Mu-Law Compressor	4-329
Mu-Law Expander	4-330
Multipath Rayleigh Fading Channel	4-331
OQPSK Demodulator Baseband	4-334
OQPSK Demodulator Passband	4-336
OQPSK Modulator Baseband	4-339
OQPSK Modulator Passband	4-342
Phase-Locked Loop	4-345
PM Demodulator Baseband	4-348
PM Demodulator Passband	4-350
PM Modulator Baseband	4-352
PM Modulator Passband	4-353
PN Sequence Generator	4-355
Poisson Int Generator	4-358
Puncture	4-360
QPSK Demodulator Baseband	4-362
QPSK Modulator Baseband	4-364
Quantizer Decode	4-367

Random Deinterleaver	4-368
Random-Integer Generator	4-369
Random Interleaver	4-372
Rayleigh Noise Generator	4-373
Rectangular QAM Demodulator Baseband	4-376
Rectangular QAM Demodulator Passband	4-379
Rectangular QAM Modulator Baseband	4-383
Rectangular QAM Modulator Passband	4-387
Rician Fading Channel	4-391
Rician Noise Generator	4-394
Sampled Quantizer Encode	4-397
Scrambler	4-399
SSB AM Demodulator Baseband	4-401
SSB AM Demodulator Passband	4-403
SSB AM Modulator Baseband	4-405
SSB AM Modulator Passband	4-408
Triggered Read From File	4-411
Triggered Write to File	4-414
Uniform Noise Generator	4-416
Viterbi Decoder	4-419
Voltage-Controlled Oscillator	4-424
Windowed Integrator	4-426

Preface

What Is the Communications Blockset?xii
Related Products	xiii
Using This Guidexv
Expected Backgroundxv
Organization of the Document	xvi
Configuration Information	xvii
Using the Blockset	xvii
Technical Conventions	xviii
Scalars, Vectors, and Matrices	xviii
Frame-Based and Sample-Based Signals	xviii
Typographical Conventionsxx

What Is the Communications Blockset?

The Communications Blockset is a collection of Simulink® blocks designed for research, development, system design, analysis, and simulation in the communications area. You can use the blockset's ready-to-use blocks directly, or you can easily modify them to implement your own methods and algorithms.

Blocks in this product can model various processes within communication systems, including:

- Signal generation
- Source coding
- Error-control coding
- Interleaving
- Modulation/demodulation
- Transmission along a channel
- Synchronization

Related Products

The MathWorks provides several products that are especially relevant to the kinds of tasks you can perform with the Communications Blockset. They are listed in the table below. In particular, the Communications Blockset *requires* these products:

- MATLAB®
- Simulink
- Signal Processing Toolbox
- Communications Toolbox
- DSP Blockset

For more information about any of these products, see either:

- The online documentation for that product, if it is installed or if you are reading the documentation from the CD
- The MathWorks Web site, at <http://www.mathworks.com>; see the “products” section

Note The toolboxes listed below all include functions that extend MATLAB’s capabilities. The blocksets all include blocks that extend Simulink’s capabilities.

Product	Description
CDMA Reference Blockset	Simulink block libraries for the design and simulation of the IS-95A wireless communications standard
Communications Toolbox	MATLAB functions for modeling the physical layer of communications systems

Product	Description
DSP Blockset	Simulink block libraries for the design, simulation, and prototyping of digital signal processing systems
Real-Time Workshop [®]	Tool that generates customizable C code from Simulink models and automatically builds programs that can run in real time in a variety of environments
Signal Processing Toolbox	Tool for algorithm development, signal and linear system analysis, and time-series data modeling
Simulink	Interactive, graphical environment for modeling, simulating, and prototyping dynamic systems
Stateflow [®]	Tool for graphical modeling and simulation of complex control logic

Using This Guide

This guide describes how to use the Communications Blockset to simulate communication systems. It contains tutorial information categorized by type of communication process, as well as a reference entry for each block in the blockset.

Expected Background

This guide assumes that you already have background knowledge in the subject of communications. If you do not yet have this background, then you can acquire it using a standard communications text or the books listed in one of this guide's sections whose titles begin with "Selected Bibliography."

If You Are a New User

Start with "Getting Started with the Communications Blockset", which describes an example in detail. Then read those parts of "Using the Communications Blockset" that address the functionality that concerns you. When you find out which blocks you want to use, refer to those parts of "Block Reference" that describe those blocks.

If You Are an Experienced User

The block reference descriptions in "Block Reference" are probably the most relevant parts of this guide for you. Each reference description includes a complete explanation of the block's parameters and operation. Many reference descriptions also include examples, a description of the block's algorithm, and references to additional reading material.

You might also want to browse through "Getting Started with the Communications Blockset" and "Using the Communications Blockset" based on your interests or needs.

Organization of the Document

This chapter introduces the Communications Toolbox and some of its important terminology. The table below summarizes the contents of the remaining chapters.

Chapter	Description
“Getting Started with the Communications Blockset”	Discusses an example model in detail to help you begin learning about the blockset
“Using the Communications Blockset”	Surveys the major libraries of the blockset and discusses their capabilities and features
“Function Reference”	Contains reference entries for the small number of functions in the blockset
“Block Reference”	Shows the contents of each library of the blockset and contains reference entries for all blocks

Configuration Information

To determine if the Communications Blockset is installed on your system, type

```
ver
```

at the MATLAB prompt. MATLAB displays information about the version of MATLAB you are running, including a list of installed add-on products and their version numbers. Check the list to see if the Communications Blockset appears.

For information about installing the blockset, see the *MATLAB Installation Guide* for your platform.

Note For the most up-to-date information about system requirements, see the system requirements page, available in the support area of the MathWorks Web site (<http://www.mathworks.com/support>).

Using the Blockset

To open the Communications Blockset, type

```
commlib
```

at the MATLAB prompt.

Double-click on any icon in the main Communications Blockset window to open the library that the icon represents.

Technical Conventions

This section discusses the terminology that this document uses to describe the signal types that the Communications Blockset supports. To learn how the blockset processes each kind of signal, see “Signal Support” on page 2-3.

Scalars, Vectors, and Matrices

This document uses the unqualified words *scalar* and *vector* in ways that emphasize a signal’s number of elements, not its strict dimension properties:

- A *scalar* signal is one that contains a single element. The signal could be a one-dimensional array with one element, or a matrix of size 1-by-1.
- A *vector* signal is one that contains one or more elements, arranged in a series. The signal could be a one-dimensional array, a matrix that has exactly one column, or a matrix that has exactly one row. The number of elements in a vector is called its length or, sometimes, its width.

In cases when it is important for a description or schematic to distinguish among different types of scalar signals or different types of vector signals, this document mentions the distinctions explicitly. For example, the terms one-dimensional array, column vector, and row vector distinguish among three types of vector signals.

The *size* of a matrix is the pair of numbers that indicate how many rows and columns the matrix has. The *orientation* of a two-dimensional vector is its status as either a row vector or column vector. A one-dimensional array has no orientation.

A matrix signal that has more than one row and more than one column is called a *full matrix* signal.

Frame-Based and Sample-Based Signals

In Simulink, each matrix signal has a “frame attribute” that declares the signal to be either frame-based or sample-based, but not both. (A one-dimensional array signal is always sample-based, by definition.) Simulink indicates the frame attribute visually by using a double connector line in the model window instead of a single connector line. In general, Simulink interprets frame-based and sample-based signals as follows:

- A frame-based signal in the shape of an M-by-1 (column) matrix represents M successive samples from a single time series.
- A frame-based signal in the shape of a 1-by-N (row) matrix represents a sample of N independent channels, taken at a single instant in time.
- A sample-based matrix signal might represent a set of bits that collectively represent an integer, or a set of symbols that collectively represent a code word, or something else *other than* a fragment of a single time series.

Typographical Conventions

This guide uses some or all of these conventions.

Item	Convention to Use	Example
Example code	Monospace font	To assign the value 5 to A, enter <code>A = 5</code>
Function names/syntax	Monospace font	The cos function finds the cosine of each array element. Syntax line example is <code>MLGetVar ML_var_name</code>
Keys	Boldface with an initial capital letter	Press the R eturn key.
Literal strings (in syntax descriptions in Reference chapters)	Monospace bold for literals.	<code>f = freqspace(n, 'whole')</code>
Mathematical expressions	Variables in <i>italics</i> Functions, operators, and constants in standard text.	This vector represents the polynomial $p = x^2 + 2x + 3$
MATLAB output	Monospace font	MATLAB responds with <code>A =</code> <code>5</code>
Menu names, menu items, and controls	Boldface with an initial capital letter	Choose the F ile menu.
New terms	<i>Italics</i>	An <i>array</i> is an ordered collection of information.
String variables (from a finite list)	<i>Monospace italics</i>	<code>sysc = d2c(sysd, 'method')</code>

Getting Started with the Communications Blockset

The Example Model	1-3
Overview of the Simulation	1-3
Building the Model	1-5
Exploring the Model	1-9
 Components of the Example	 1-11
Creating Random Binary Data	1-11
Encoding Using a Convolutional Code	1-12
Unbuffering to Convert Vectors to Scalars	1-14
Modulating the Encoded Messages	1-16
Transmitting Along a Noisy Channel	1-16
Mapping the Received Data	1-18
Buffering to Convert Scalars to Vectors	1-20
Decoding the Convolutional Code	1-21
Computing the Error Rate	1-24
Displaying the Error Rate	1-26
Other Blocks	1-26
 Learning More About the Example	 1-27
Modifying the Model	1-27
For Further Study	1-28

This chapter describes a particular example in detail, to help you get started using the Communications Blockset. The description here assumes very little prior knowledge of MATLAB or Simulink. It assumes that you have a basic knowledge about communications subject matter. More specialized knowledge about convolutional coding might be useful for the two parts marked “Technical,” but is *not* essential for using the example to learn about the Communications Blockset environment.

This chapter:

- Gives an overview of the model
- Discusses how you can explore the model in Simulink before reading about its components
- Describes the components of the model in detail
- Discusses how you can modify the model after you understand how it works
- Lists sources of further information

The detailed component descriptions also include three digressions about:

- Signal sizes (page 1-12)
- Sample times (page 1-15)
- Data types (page 1-16)

While these digressions relate to specific components of the example model, they also point out *general* model-building and model-debugging techniques that can help you use Simulink more effectively.

The Example Model

The figure below shows the example model. You can open it by typing `commblksgettingstarted` at the MATLAB prompt.

Tip In this document, you can click on blocks in the figure above to see more information about how they behave within the example.

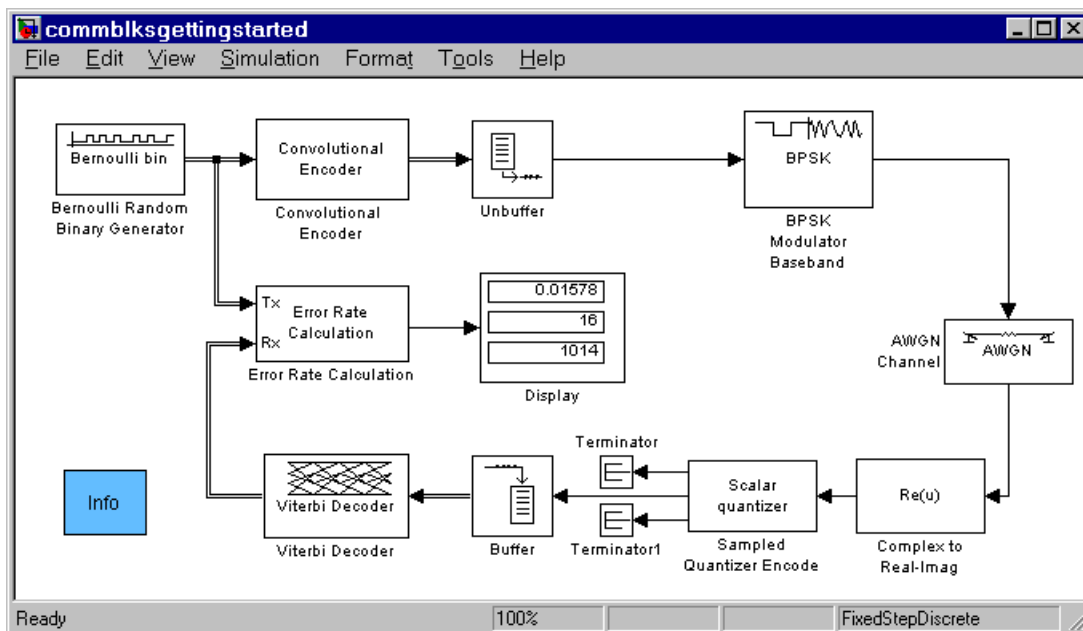


Figure 1-1: Example Model

Overview of the Simulation

This example simulation starts by creating a random binary message signal. The simulation encodes the message into a convolutional code, modulates the code using the binary phase shift keying (BPSK) technique, and adds white Gaussian noise to the modulated data in order to simulate a noisy channel.

Then, the simulation decodes the convolutional code while trying to correct as many noise-induced errors as possible. The decoding also involves some intermediate steps to prepare the received data for the decoding block. Finally, the simulation compares the decoded information to the original message signal in order to compute and display an error rate.

The table below indicates which blocks from the Communications Blockset appear in the model, the order they appear in the model, and the purpose each one serves.

Communications Blockset Block	Purpose in Example
Bernoulli Random Binary Generator	Create random bits to use as message.
Convolutional Encoder	Encode message using the convolutional coding technique.
BPSK Modulator Baseband	Modulate encoded message to prepare for transmission.
AWGN Channel	Transmit data, adding random numbers to simulate a noisy channel.
Sampled Quantizer Encode	Map received data to appropriate three-bit values to prepare for soft-decision decoding.
Viterbi Decoder	Decode the convolutional code using the Viterbi algorithm.
Error Rate Calculation	Compute proportion of discrepancies between original and recovered messages.

The model also uses some blocks from Simulink and the DSP Blockset:

- Unbuffer (DSP Blockset)
- Complex to Real-Imag (Simulink)
- Terminator (Simulink)
- Buffer (DSP Blockset)
- Display (Simulink)

Building the Model

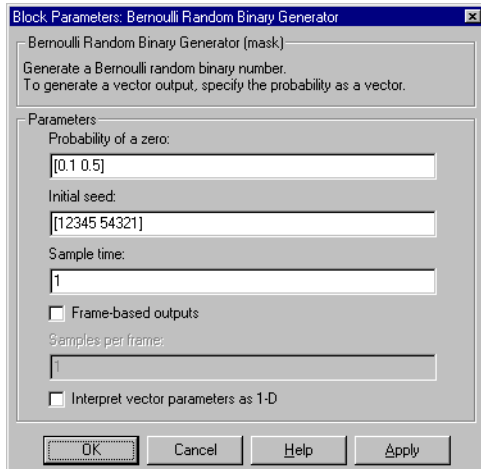
This section helps you build the model starting from a blank model window. If you prefer to open the prebuilt model, then type `commblksgettingstarted` at the MATLAB prompt and skip ahead to “Exploring the Model” on page 1-9.

Part 1: Placing the First Block

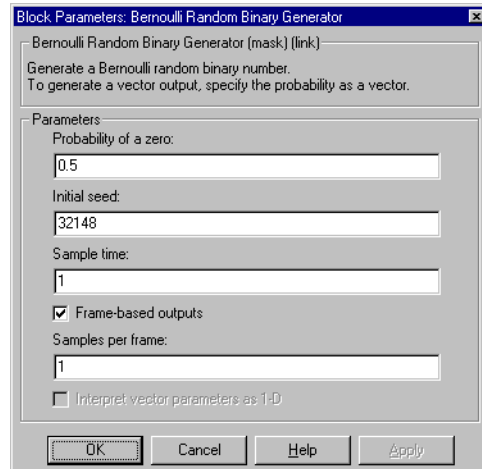
To start building the model, follow these steps:

- 1 Type `commlib` at the MATLAB prompt. This opens the Communications Blockset main library.
- 2 From the library's **File** menu, select **New** and then select **Model**. This opens a new model window called `untitled`. You will build the example model in this window.
- 3 From the model window's **File** menu, select **Save**. Choose an appropriate location and filename for the model you are about to build. You should save the model frequently while you are editing it, to avoid losing any work.
- 4 In the Communications Blockset main library, double-click on the **Comm Sources** icon. This opens the **Comm Sources** library.
- 5 In the **Comm Sources** library, find the icon for the **Bernoulli Random Binary Generator** block. Drag it into the model window.
- 6 In the model window (not the **Comm Sources** library window), double-click on the **Bernoulli Random Binary Generator** icon. This opens the block's parameter dialog box, also called its *mask*.
- 7 In the mask, type new values in the parameter fields to change the default parameter values to the ones shown in the right image below.

Default Parameters



Desired Parameters for the Example



- 8 Click on the **OK** button in the mask.

You have now placed and configured the first block for this example model.

Part 2: Placing Other Blocks

This section tells how to find and configure the other blocks required for the example model. First open the main libraries of the products that contain those blocks:

- To open the main DSP Blockset library, type `dspl i b` at the MATLAB prompt.
- To open the main Simulink library, type `si mul i nk3` at the MATLAB prompt.

Now, the basic procedure for placing blocks is similar to the procedure you used for the Bernoulli Random Binary Generator block:

- 1 From the product's main library, navigate to the library or sublibrary where the block resides.
- 2 Drag the desired block into the model window.

- 3 Double-click on the block in the model window to open its mask.
- 4 Change the appropriate parameters.
- 5 Click on the mask's **OK** button.

Apply this basic procedure to the blocks listed below. For now, just place each block anywhere within your model window. The next section gives tips for connecting the blocks to each other.

Below are the blocks that you should gather and configure in your model window. Each bullet lists the name and library location of the block, while each subbullet indicates how to change the parameters from their default values.

From the Communications Blockset Library.

- Convolutional Encoder, from the Convolutional sublibrary of the Channel Coding library. Use default parameter values.
- BPSK Modulator Baseband, from the PM sublibrary of the Digital Baseband sublibrary of the Modulation library. Use default parameter values.
- AWGN Channel, from the Channels library
 - Set **Initial seed** to 123456
 - Set **Es/No** to - 1
 - Set **Symbol period** to . 5
- Sampled Quantizer Encode, from the Source Coding library
 - Set **Quantization partition** to [- . 75 - . 5 - . 25 0 . 25 . 5 . 75]
 - Set **Quantization codebook** to [7 6 5 4 3 2 1 0]
 - Set **Input signal vector length** to 1
 - Set **Sample time** to - 1
- Viterbi Decoder, from the Convolutional sublibrary of the Channel Coding library
 - Set **Decision type** to **Soft Decision**
 - Set **Number of soft decision bits** to 3
 - Set **Traceback depth** to 48

- Error Rate Calculation, from the Comm Sinks library
 - Set **Receive delay** to 49
 - Set **Output data to Port**

From the DSP Blockset Library.

- Unbuffer, from the Buffers sublibrary of the Signal Management library.
Use the default parameter value.
- Buffer, from the Buffers sublibrary of the Signal Management library
 - Set **Output buffer size** to 2

From the Simulink Library.

- Complex to Real-Imag, from the Math library
 - Set **Output to Real**
- Two copies of Terminator, from the Signals & Systems library
- Display, from the Sinks library
 - Use default parameter values, but drag a corner of the icon to make it three times as tall

Connecting the Blocks

Once you have all the blocks in your model window, you can connect them so that your model window looks like “Example Model” on page 1-3 (except for the Info block that appears in that figure). Also, from the model window’s **Simulation menu**, choose **Simulation parameters**; then in the **Simulation Parameters** dialog box, set **Stop time** to `inf`.

Here are some tips for connecting blocks:

- To connect the output port of one block to the input port of another block, position the pointer over the first block’s output port and drag the pointer to the second block’s input port.
- To add a branch to an existing connection line (for example, to connect the Bernoulli Random Binary Generator block to *both* the Convolutional Encoder block and the Error Rate Calculation block), first position the pointer on the line where you want the branch to start. Then using the right mouse button, drag the pointer to the place where you want the branch to

end. As an alternative to using the right mouse button, you can also use the left mouse button while holding down the **Ctrl** key.

- To delete a connection line, first select it by positioning the cursor along the line and pressing the mouse button. Then select **Cut** from the model window's **Edit** menu.
- To undo the addition or deletion of a block or line, choose **Undo** from the model window's **Edit** menu. You can also reverse the effect of an **Undo** command by using the **Redo** option from the model window's **Edit** menu.
- To reverse the orientation of a block's icon, select the block and choose **Flip block** from the model window's **Format** menu. To rotate a block's icon, select the block and choose **Rotate block** from the model window's **Format** menu.

Additional information about modelbuilding is in the section, "Creating a Model," in the *Using Simulink* guide.

Exploring the Model

Once the model is open, you can explore it in several ways. You can begin to explore even before you run the simulation:

- "Read" the block diagram starting with the Bernoulli Random Binary Generator block in the upper left corner, following the arrows, and ending with the Display block in the center.

Note that the block marked "Info" does not function during the simulation but rather links to the HTML version of this documentation.

- Double-click on any block to see its parameter dialog box. This dialog box briefly describes the block, shows its parameter values if the block has parameters, allows you to change any of the parameter values, and also includes a **Help** button that links to the detailed HTML reference page for the block.
- To see the blockset library in which any Communications Blockset block resides, right-click on the block, select **Link options**, and then select **Go to library block**. This shows you where to find the block if you later want to use it in your own model. The **Link options** function is inactive for built-in Simulink blocks, such as the Display block.

Running the Simulation

Run the simulation by selecting **Start** from the model window's **Simulation** menu. While the simulation runs, the bottom bar of the model window displays the time, T, and the Display block in the center of the block diagram displays three numbers that represent the simulation's error rate information (explained below). Once it starts, the simulation runs until you select **Stop** from the model window's **Simulation** menu.

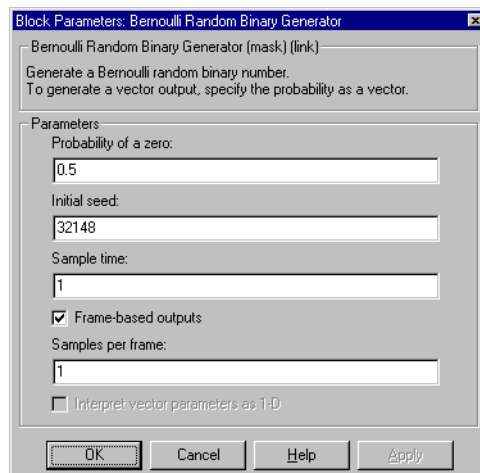
Components of the Example

This section discusses the purpose, behavior, and relevant parameters of each block within the example model. Except for two noncomputational blocks discussed at the end, this section covers blocks in the order in which they process data in the simulation.

Creating Random Binary Data

The Bernoulli Random Binary Generator block, in the Comm Sources library, produces the message information for this model. The output of this block is what the other components of the model encode, modulate, transmit, and decode in turn.

Double-click on the Bernoulli Random Binary Generator block in the model window to open the parameter mask dialog box shown below.



Since the **Sample time** parameter is 1 second, the block generates one binary number each second. Since the **Probability of a zero** parameter is 0.5, the block generates the bits so that 0 and 1 are equally probable. The **Initial seed** parameter initializes the random number generator; if you change the **Initial seed** parameter, then the block generates a different random sequence.

Because the **Frame-based outputs** check box is checked, the block produces a frame-based scalar signal instead of a sample-based scalar signal. This is done

to accommodate the functionality of the Unbuffer and Buffer blocks in a different part of the model, and illustrates one of several possible ways to handle signals appropriately throughout the model as a whole. When you run the simulation or update the diagram, notice that the connector line that leads out of the Bernoulli Random Binary Generator block is a double line instead of a single line. This double line indicates a frame-based signal.

Encoding Using a Convolutional Code

The Convolutional Encoder block, in the Convolutional sublibrary of the Channel Coding library, receives the messages from the Bernoulli Random Binary Generator block and encodes them into codewords.

While the message data is a scalar bit stream, the encoded data is a stream of binary length-two vectors. These signal sizes are compatible with the structure of the particular convolutional code and with the corresponding parameter configuration of the Convolutional Encoder block.

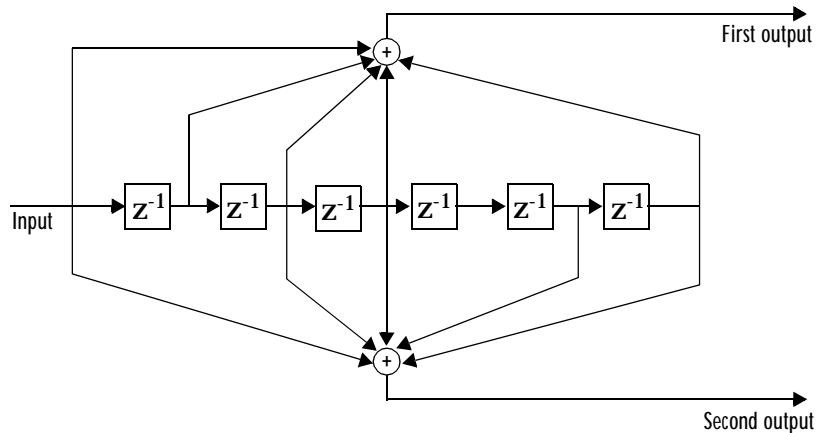
Digression: Exploring Signal Sizes

This section explains a technique for gathering information about the sizes of signals in a model. This technique can be useful for determining what parameters you should use for a block or for diagnosing problems in a model.

To check the sizes of signals in the model, use the **Signal dimensions** feature from the model window's **Format** menu. In this example, when the signal dimension display is on, the connector line that leads out of the Convolutional Encoder block has the annotation [2x1] above it because the signal is a 2-by-1 matrix signal. The connector line that leads into the Convolutional Encoder block has no annotation above it because it is a scalar. The situation is the opposite for the Viterbi Decoder block.

Defining the Convolutional Code (Technical)

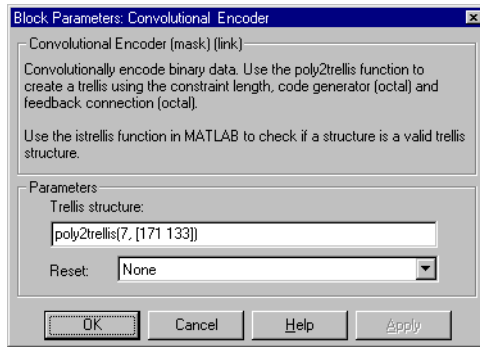
The feedforward convolutional encoder in this example is depicted below.



Each summing node represents modulo-2 addition. Each box marked z^{-1} represents a memory register that holds the input values from previous sample times. Since there are six memory registers, the output at a given time depends on seven input values, including the current one. Thus the *constraint length* of the code is 7. Since the code has one input and two outputs, the code rate is 1/2.

A pair of octal numbers called the *code generator* indicates the connections from the memory registers to the modulo-2 summing nodes. The pair [171 133] describes the encoder in the figure.

The **Trellis structure** parameter in the Convolutional Encoder block tells the block which code to use when processing data. In this case, the `poly2trellis` function, in the Communications Toolbox, converts the constraint length and the pair of octal numbers into a valid trellis structure that the block uses in its processing.



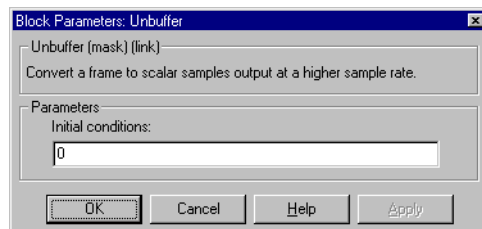
Computing the Code Generator. The code generator is a 1-by-2 matrix of octal numbers because the encoder has one input and two outputs. The first element in the matrix indicates which input values contribute to the first output, and the second element in the matrix indicates which input values contribute to the second output.

For example, the first output in the encoder diagram is the modulo-2 sum of the rightmost and the four leftmost elements in the diagram's array of input values. The seven-digit binary number 1111001 captures this information, and is equivalent to the octal number 171. The octal number 171 thus becomes the first entry of the code generator matrix. Here, each triplet of bits uses the leftmost bit as the most significant bit.

The second output corresponds to the binary number 1011011, which is equivalent to the octal number 133. The code generator is therefore [171 133].

Unbuffering to Convert Vectors to Scalars

After the Convolutional Encoder block encodes the data, the goal is to modulate the codewords. However, the codewords are vectors of length two, while this model chooses to modulate, transmit, and quantize only scalar data. Thus an intermediate step converts the vector codewords into a scalar signal. The Unbuffer block, in the DSP Blockset, performs this intermediate step. To check the sizes of the input and output of the Unbuffer block, try the technique mentioned in "Digression: Exploring Signal Sizes" on page 1-12.



The Unbuffer block is a multirate block, which means that its input sample rate differs from its output sample rate. In this case, the Unbuffer block receives one length-two codeword every second and outputs a scalar every *half* second.

Digression: Exploring Sample Times

Because sample times are important in communication system simulation, this section explains two techniques for gathering information about the sample times of components of a model. These techniques can be useful for determining what parameters you should use for a block or for diagnosing problems in a model.

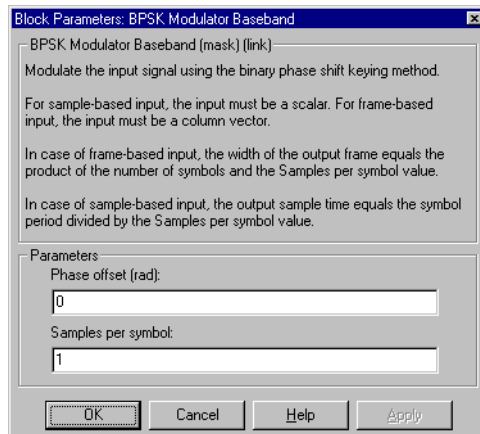
Exact Sample Time of a Signal. To check the sample time of any signal in the model, attach a Probe block to the signal's line. The Probe block resides in Simulink's Signals & Systems library. Simply drag a Probe block into the model window, attach its input port to the signal line you want to examine, and then update the diagram by selecting **Update diagram** from the model window's **Edit** menu. The first number after the Ts or Tf notation on the Probe block's icon is the length of time between updates of that signal. (In the case of frame-based signals, the Tf notation indicates that the number shown is the period of the entire frame signal, not the period of each row of the frame.) For more information, see the reference entry for the Probe block in the Simulink documentation set. In this model, a Probe block attached to the line that leads out of the Unbuffer block shows a sample time of 0.5 second.

Relative Sample Times in the Model. Another way to explore sample times is to turn on the sample time coloring feature for the whole model. Use the **Sample time colors** option in the model window's **Format** menu to toggle the coloring feature on and off. If sample time coloring is on, then blocks with the same sample time have the same color. In this example, the Unbuffer block becomes yellow because it is a multirate block. Also, the blocks before and after the

Unbuffer block have different colors because they have different sample times. For more information about sample times, refer to Simulink documentation.

Modulating the Encoded Messages

The BPSK Modulator Baseband block, in the Digital Baseband Modulation sublibrary of the Modulation library, modulates the scalar stream of coded data. It simulates binary phase shift keying (BPSK) modulation. The output of the BPSK Modulator Baseband block is a complex Simulink signal, even though the BPSK-modulated values happen to be real.



Digression: Exploring Data Types

To check the data types of signals in a model, use the **Port data types** feature from the model window's **Format** menu. In this example, when this feature is on, the connector line that leads out of the BPSK Modulator Baseband block has a label "double (c)" above it to indicate that it is a complex double-precision signal. By contrast, the connector line that leads into the BPSK Modulator Baseband block is labeled "double" to indicate that it is a real double-precision signal.

Transmitting Along a Noisy Channel

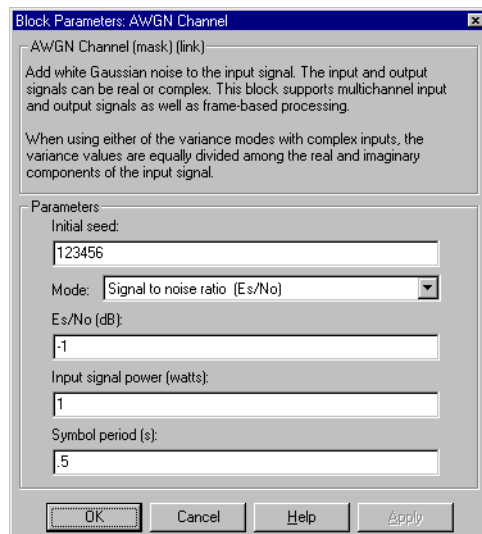
After modulation, the data is ready for transmission. The AWGN Channel block, in the Channels library, simulates a noisy channel by adding white

Gaussian noise to the data stream. The AWGN Channel block uses a Gaussian distribution whose variance is determined using these mask parameters:

- **Es/No**, which is - 1 decibel in this example
- **Input signal power**, which is 1 watt because BPSK modulation produces values of -1 and 1
- **Symbol period**, which is .5 second

The AWGN Channel block can compute the variance in other ways as well, but this example chooses this mode by setting the **Mode** parameter to **Signal to noise ratio (Es/No)**.

Like the Bernoulli Random Binary Generator block, the AWGN Channel block requires an **Initial seed** parameter that initializes the random number generator. If you change the **Initial seed** parameter, then the block generates a different random sequence. The **Initial seed** parameters of the AWGN Channel block and the Bernoulli Random Binary Generator block do not need to match.



Note The input to the AWGN Channel block is a complex Simulink signal, not a real one (See “Digression: Exploring Data Types” on page 1-16.). This causes the AWGN Channel block to add complex noise, dividing the calculated variance equally between the real and imaginary components.

Mapping the Received Data

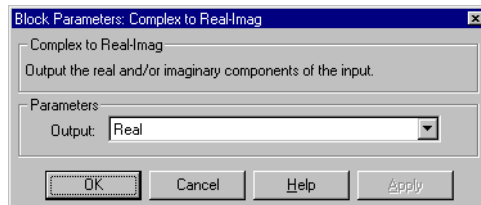
The received data, that is, the output of the AWGN Channel block, consists of complex numbers that are close to -1 and 1. In order to reconstruct the original binary message, the receiver part of the model must decode the convolutional code. Although the Viterbi Decoder block is designed for convolutional decoding, it expects its input data to have a particular format. The purpose of the mapping task is to transform the output of the AWGN Channel block into a format that the Viterbi Decoder block can interpret properly.

Specifically, the Viterbi Decoder block is configured here to process integer input values between 0 and 7. The mapping procedure includes these two tasks:

- 1 Convert the received data signal to a real signal by removing its imaginary part. It is reasonable to assume that the imaginary part of the received data does not contain essential information, because the imaginary part of the transmitted data is zero (ignoring small roundoff errors) and because the channel noise is not very powerful.
- 2 Map the resulting real signal to an integer between 0 and 7, to prepare for the soft-decision algorithm described in the next section.

Converting to Real Data

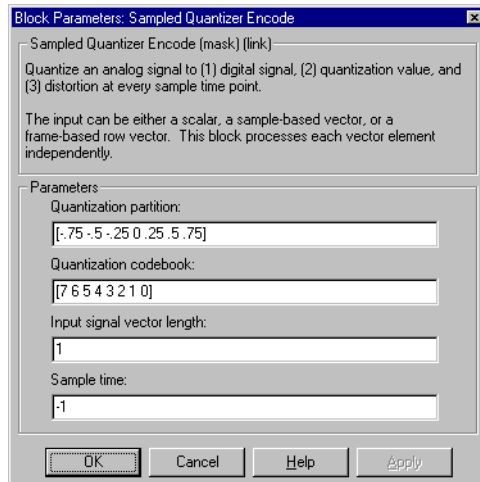
The Complex to Real-Imag block, in Simulink’s Math library, performs step 1 above. Since its **Output** parameter is set to **Real**, the block outputs only the real part of its input.



Mapping to Decision Values

The Sampled Quantizer Encode block, in the Source Coding library, performs step 2 above. This block classifies each real input value based on whether it is greater or less than the partition values in the **Quantization partition** parameter. The block then outputs a value from the **Quantization codebook** parameter that depends on the classification.

As the section “Defining the Decoding Process (Technical)” on page 1-22 explains, the Viterbi Decoder requires input values that are integers between 0 and 2^3-1 in order to decode using three-bit soft decisions. Therefore, the **Quantization codebook** parameter is a vector containing the eight integers between 0 and 7. Since the input to the Sampled Quantizer Encode block is close to the values -1 and 1, it is reasonable to select the seven partition values to be -.75, -.5, -.25, 0, .25, .5, and .75. This means that the first classification group consists of numbers less than or equal to -.75, the second classification group consists of numbers between -.75 and -.5 (including -.5), and so on, until the eighth classification group consists of numbers greater than .75. Finally, the elements of the **Quantization codebook** and **Quantization partition** vectors are ordered so that an input value of -1 maps to an output value of 7 and an input value of 1 maps to an output value of 0. The combination of this mapping and the Viterbi Decoder block’s decision mapping reverses the BPSK modulation that the BPSK Modulator Baseband block performs on the transmitting side of this model.

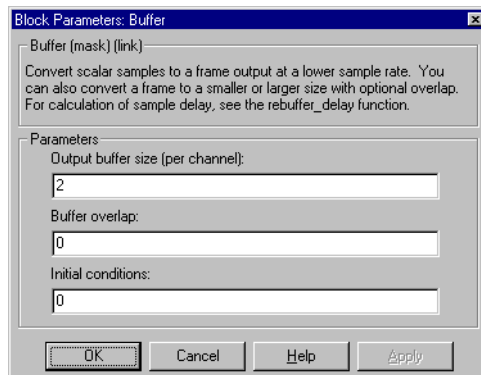


Buffering to Convert Scalars to Vectors

The output of the Sampled Quantizer Encode block has values tailored to the Viterbi Decoder block's expectations. However, it is not yet ready for decoding because it is a scalar, whereas the convolutional code in this example uses codewords that are vectors of length two. The Buffer block, from the DSP Blockset, converts the scalar output of the Sampled Quantizer Encode block into a length-two vector signal. The output of the Buffer block is suitable input for the Viterbi Decoder block.

The Buffer block's outputs have length two because its **Output buffer size** parameter is 2. The block combines its inputs over the first two sample times into a single vector output; then it combines its input over the third and fourth sample times into the subsequent single vector output, and so on.

Like the Unbuffer block, the Buffer block is a multirate block. The period of the Buffer block's output is twice as long as the period of its input. For more about how to check sample times of blocks, see "Digression: Exploring Sample Times" on page 1-15.

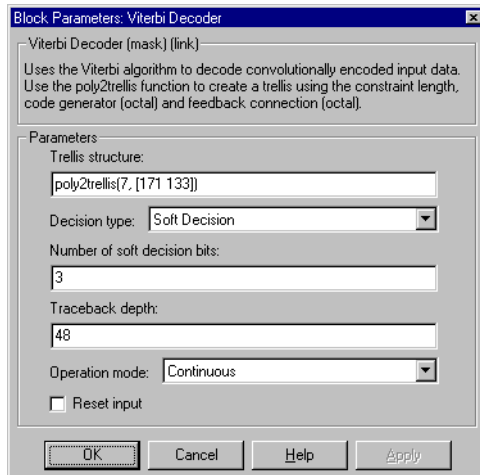


Decoding the Convolutional Code

Now that the received data is properly mapped to length-two vectors of 3-bit decision values, the Viterbi Decoder block can decode it. The Viterbi Decoder block resides in the Convolutional sublibrary of the Channel Coding library.

The **Trellis structure** parameter in the Viterbi Decoder block defines the encoder and therefore matches the corresponding parameter in the model's Convolutional Encoder block. For more information about this parameter, see “Encoding Using a Convolutional Code” on page 1-12.

Other parameters in the Viterbi Decoder block are specific to the decoding process. The block uses soft decisions with 2^3 different input values because the **Decision type** parameter is **Soft Decision** and the **Number of soft decision bits** parameter is 3.



Defining the Decoding Process (Technical)

This section elaborates on some more advanced aspects of the decoding in this example. It discusses how the Viterbi Decoder block interprets its inputs, how the interpretation influences the mapping done by the blocks that precede the Viterbi Decoder block, and what the **Traceback depth** parameter means.

Soft-Decision Interpretation of Data. When the **Decision type** parameter is set to **Soft Decision**, the Viterbi Decoder block requires input values between 0 and $2^b - 1$, where b is the **Number of soft decision bits** parameter. The block interprets 0 as the most confident decision that the codeword bit is a zero and interprets $2^b - 1$ as the most confident decision that the codeword bit is a one. The values in between these extremes represent less confident decisions.

The table below lists the interpretations of the eight possible input values for this example.

Table 1-1: Decision Values for 3-Bit Soft Decisions

Decision Value	Interpretation
0	Most confident 0
1	Second most confident 0
2	Third most confident 0
3	Least confident 0
4	Least confident 1
5	Third most confident 1
6	Second most confident 1
7	Most confident 1

How Decoder’s Interpretation Influences Choice of Mapping. The table above helps explain the design of the mapping process described in “Mapping the Received Data” on page 1-18. For example, if the original message contains a 1, then the Viterbi Decoder block makes a correct decision if and only if it receives an input between 4 and 7. On the other hand, the 1 in the message gets mapped to -1 by the BPSK Modulator Baseband block and to a number near -1 by the AWGN Channel block. This means that the Sampled Quantizer Encode block should be configured so as to map input numbers near -1 to output numbers between 4 and 7. As the **Quantization partition** and **Quantization codebook** parameters for the Sampled Quantizer Encode block indicate, it maps input numbers less than 0 to output numbers between 4 and 7. The schematic below suggests how knowledge about the modulator, channel, and decoder decisions (solid arrows) can help you design intermediate steps like the mapping procedure (dashed arrow).

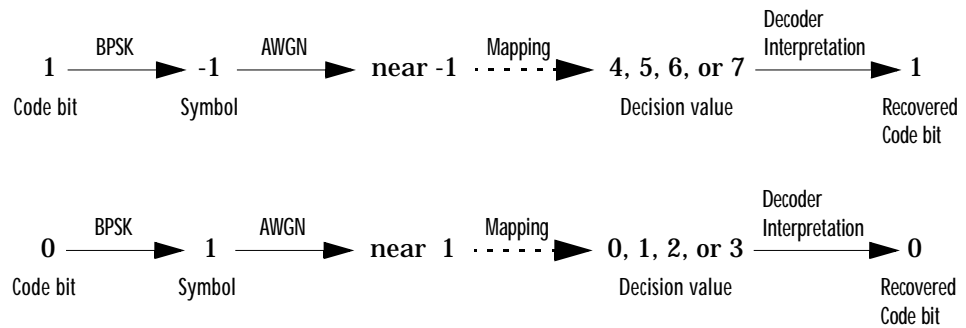


Figure 1-2: Processing and Recovery of Code Bits

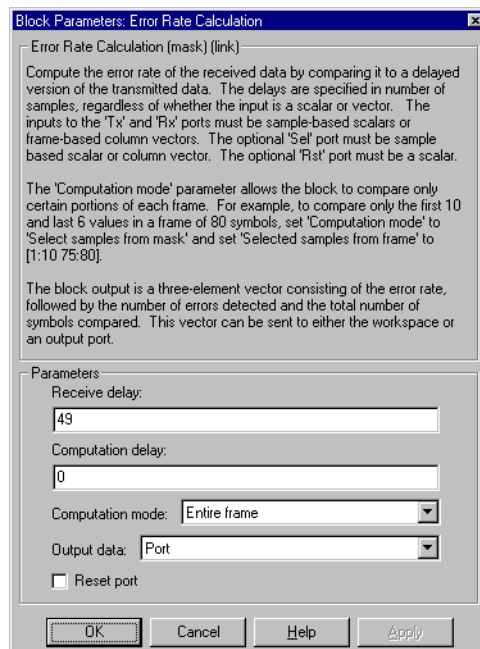
Traceback Depth Parameter. The **Traceback depth** parameter in the Viterbi Decoder block represents the length of the decoding delay. Typically, people aim for a traceback depth of about five or six times the constraint length, which would be 35 or 42 in this example. However, some hardware implementations offer options of 48 and 96. This example chooses 48 because that is closer to the targets (35 and 42) than 96 is.

Computing the Error Rate

The Error Rate Calculation block, in the Comm Sinks library, compares the original message from the Bernoulli Random Binary Generator block with the recovered message from the Viterbi Decoder block, and produces error statistics. The Error Rate Calculation block produces a length-three vector whose elements are:

- The error rate, which is the quotient of the next two quantities below
- The total number of errors, that is, comparisons between unequal elements
- The total number of comparisons that the block made

Since the block's **Output data** parameter is **Port**, the block sends its error statistics to the output port. At the output port, a Display block receives and displays the error statistics.



Delay in Received Data

One subtlety of the Error Rate Calculation block's configuration is the **Receive delay** parameter. This parameter is nonzero because a given message bit and its corresponding recovered bit are separated in time by a nonzero amount of simulation time. The **Receive delay** parameter tells the block which elements of its input signals to compare when checking for errors.

Tip If you build your own model using the Error Rate Calculation block and get an error rate close to .5 for binary data, then you should check whether the **Receive delay** value is the correct delay for the model.

In this case, the **Receive delay** value is 49 samples, which is one more than the **Traceback depth** value (48) in the Viterbi Decoder block. The extra one-sample delay comes from the initial delay in the Buffer block. Because the

Buffer block must collect two scalar samples before it can output one vector, its first meaningful output occurs at time 1 second, not time 0.

Displaying the Error Rate

The Display block, in Simulink's Sinks library, receives the length-three output of the Error Rate Calculation block and displays it while the simulation runs. The three fields represent the error rate, the total number of errors, and the total number of comparisons that the Error Rate Calculation block made during the simulation. The total number of comparisons depends on how long you let the simulation run. The simulation runs until you end it.

Other Blocks

The Terminator and Info blocks do not perform a function during the simulation, yet they serve other purposes.

Terminator Blocks

The Terminator blocks, from Simulink's Signals & Systems library, attach to output ports whose signals are not used in the rest of the model. Simulink warns you if it finds an output port that is not attached to another block. Terminator blocks prevent warnings from occurring.

Info Block

The block marked "Info" is a customized version of Simulink's Model Info block. It is customized so that double-clicking on it displays the HTML version of this documentation in the Help browser.

Learning More About the Example

After you understand how the model works, you can learn even more about it and about how the Communications Blockset works by modifying the model or by reading other sources of information. This section suggests modifications and reference sources that might be helpful.

Modifying the Model

The table below describes some changes that you can make in the model and how to implement such changes. Changes that require a greater understanding of the blocks or techniques involved appear towards the end of the table.

Table 1-2: Ideas for Modifying the Example Model

Description of Change	Modification of Model
Change sequences of random numbers	Change the Initial seed parameter in the Bernoulli Random Binary Generator block and/or the AWGN Channel block to another positive integer.
Change distribution of random numbers in message signal	Change the Probability of a zero parameter in the Bernoulli Random Binary Generator block to another number between 0 and 1.
Change amount of noise in channel	Change the Es/No parameter in the AWGN Channel block to another real number. Alternatively, change the Mode parameter to Signal to noise ratio (SNR) or Variance from mask and set the associated SNR or Variance parameter, respectively.
Change traceback depth in decoder	Change the Traceback depth parameter in the Viterbi Decoder block to another positive integer, and also change the Receive delay parameter in the Error Rate Calculation block to that integer plus one.

Table 1-2: Ideas for Modifying the Example Model (Continued)

Description of Change	Modification of Model
Change convolutional code	<p>Change the Trellis structure parameters in both the Convolutional Encoder and Viterbi Decoder blocks. For example, change it to <code>poly2trellis(9, [753 561])</code>.</p> <p>Note that if the new code rate is not 1/2, then you might need to change other parts of the model to ensure that the signal sizes and sample times are appropriate throughout the model.</p>
Change number of soft-decision bits	<p>Change the Number of soft decision bits in the Viterbi Decoder block, and also change the Quantization partition and Quantization codebook parameters in the Sampled Quantizer Encode block. For example, change these three parameters to 2, [-.5 0 .5], and [3 2 1 0], respectively.</p> <p>More generally, the length of Quantization partition must be one less than the length of Quantization codebook, and the values in Quantization codebook must be between 0 and 2^b-1, where b is the Number of soft decision bits parameter.</p>

For Further Study

For more information about the communications techniques in this example, see a reference text such as the ones listed below. The book by Clark and Cain is particularly useful for convolutional coding.

[1] Clark, George C. Jr. and J. Bibb Cain. *Error-Correction Coding for Digital Communications*. New York: Plenum Press, 1981.

[2] Couch, Leon W. II. *Digital and Analog Communication Systems*. New York: Macmillan Publishing Company, 1990.

[3] Jeruchim, Michel C., Philip Balaban, and K. Sam Shanmugan. *Simulation of Communication Systems*. New York: Plenum Press, 1992.

[4] Sklar, Bernard. *Digital Communications: Fundamentals and Applications*. Englewood Cliffs, N.J.: Prentice-Hall, 1988.

To learn more about how to use the Communications Blockset, continue reading this guide. “Using the Communications Blockset” covers each of the core blockset libraries in turn.

To learn more about a particular block in this model, see its block reference page. Communications Blockset reference pages make up the “Block Reference” chapter in this document. If you have the model open, then you can click on the **Help** button in the block’s dialog box to display the reference page. You can also click on a Communications Blockset block name below to see its reference page:

- Bernoulli Random Binary Generator
- Convolutional Encoder
- BPSK Modulator Baseband
- AWGN Channel
- Sampled Quantizer Encode
- Viterbi Decoder
- Error Rate Calculation
- Display (Simulink)

Using the Communications Blockset

Signal Support	2-3
Communications Sources	2-6
Communications Sinks	2-12
Source Coding	2-16
Block Coding	2-27
Convolutional Coding	2-40
Interleaving	2-46
Analog Modulation	2-53
Digital Modulation	2-64
Channels	2-84
Synchronization	2-90

This chapter describes and illustrates how to implement communication techniques using the blocks in the Communications Blockset. The first section, “Signal Support,” discusses the types of signals that this blockset supports. Each subsequent section corresponds to one of the core libraries within the Communications Blockset. These sections are:

- “Communications Sources” on page 2-6
- “Communications Sinks” on page 2-12
- “Source Coding” on page 2-16
- “Block Coding” on page 2-27
- “Convolutional Coding” on page 2-40
- “Interleaving” on page 2-46
- “Analog Modulation” on page 2-53
- “Digital Modulation” on page 2-64
- “Channels” on page 2-84
- “Synchronization” on page 2-90

For descriptions of individual blocks, see their entries in Chapter 4, “Block Reference.” For background or theoretical information about communications techniques, see the works listed in the “Selected Bibliography...” sections that appear in this chapter.

Signal Support

As of Release 12, Simulink supports matrix signals in addition to one-dimensional arrays, and frame-based signals in addition to sample-based signals. This section describes how the Communications Blockset processes certain kinds of matrix and frame-based signals. To learn the terminology that this document uses to describe signal attributes, see “Technical Conventions” on page xviii.

Processing Vectors and Matrices

These rules indicate the shapes of sample-based signals that Communications Blockset blocks can process:

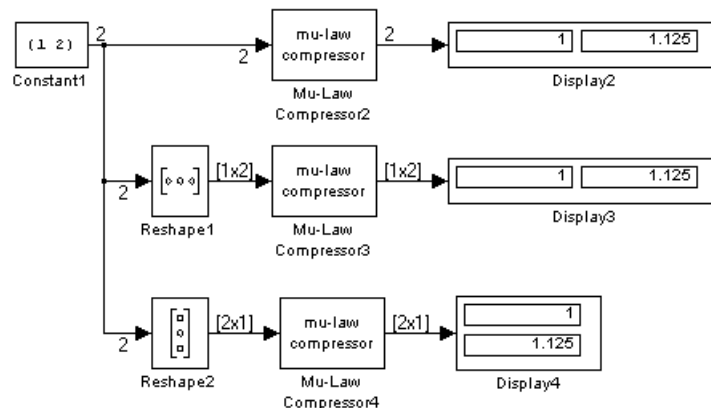
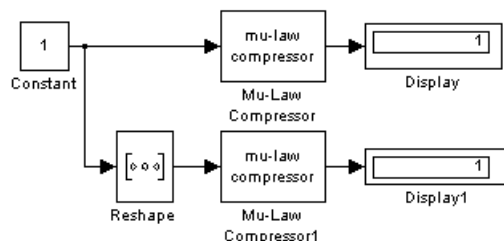
- Most blocks do not process matrix signals that have more than one row and more than one column.
- In their numerical computations, blocks that process scalars do not distinguish between one-dimensional scalars and one-by-one matrices. If the block produces a scalar output from a scalar input, then the block preserves dimension.
- If a block can process sample-based vectors, then:
 - The numerical computations do not distinguish between one-dimensional arrays, M-by-1 matrices, and 1-by-N matrices.
 - The block output preserves dimension and orientation.
 - The block treats elements of the input vector as a collection that arises naturally from the block’s operation (for example, a collection of symbols that jointly represent a codeword), or as samples from independent channels. The block does *not* assume that the elements of the input vector are successive samples from a single time series.

Some blocks process vectors but require them to be frame-based. For more information about processing frame-based signals, see “Processing Frame-Based and Sample-Based Signals” on page 2-4.

To find out whether a block processes scalar signals, vector signals, or both, refer to its entry in the reference section.

Illustrations of Scalar and Vector Processing

The figures below depict the preservation of dimension and orientation when a block processes scalars (without oversampling) and vectors. To display signal dimensions in your model, turn on the **Signal dimensions** option in the model window's **Format** menu.



Processing Frame-Based and Sample-Based Signals

All one-dimensional arrays are sample-based, but a matrix signal can be either frame-based or sample-based. A frame-based signal in the shape of an N-by-1 matrix represents a series of N successive samples from a single time series. The Communications Blockset processes some frame-based signals and is compatible with the DSP Blockset. However, the Communications Blockset omits some frame-based features, and many blocks are not specifically optimized for frame-based processing.

These rules indicate how most Communications Blockset blocks handle frame-based matrix signals:

- Most blocks do not process frame-based matrix signals that have more than one row and more than one column.
- Most blocks do not process frame-based row vectors and do not support multichannel functionality.
- Blocks that process continuous-time signals do not process frame-based inputs. Such blocks include the analog modulation blocks and the analog phase-locked loop blocks.
- Blocks for which a frame-based multichannel operation would make sense, even if the blocks do not currently support such operation, reject sample-based vectors because their interpretation is ambiguous.

Frame-based vectors, however, have an unambiguous interpretation. Blocks interpret a frame-based row vector as multiple channels at a single instant of time, and interpret a frame-based column vector as multiple samples from a single time series (that is, a single channel).

- Some blocks, such as the digital baseband modulation blocks, can produce multiple output values for each value of a scalar input signal. In such cases, a frame-based one-by-one matrix input results in a frame-based column vector output. By contrast, a sample-based scalar input results in a sample-based scalar output with a smaller sample time.

Communications Sources

Every communication system contains one or more sources. You can find sources in Simulink's Sources library, in the DSP Blockset's DSP Sources library, and in the Communication Blockset's Comm Sources library.

You can open the Comm Sources library by double-clicking on its icon in the main Communications Blockset library (commlib), or by typing

```
commsource2
```

at the MATLAB prompt.

Source Features of the Blockset

Blocks in this library can:

- Generate random or pseudorandom signals
- Generate nonrandom signals by reading from a file or by simulating a voltage-controlled oscillator (VCO)

This section describes these capabilities, considering first random and then nonrandom signals.

Random or Pseudorandom Signals

Random signals are useful for simulating noise, errors, or signal sources. Besides using built-in Simulink blocks such as the Random Number block, you can also use blocks in the Comm Sources library of the Communications Blockset to generate:

- Random bits
- Random integers
- Pseudorandom integers
- Random real numbers

This section discusses the sample time parameter, seed parameter and signal attribute parameters that are common to many random source blocks, and then discusses each category of random source.

Sample Time Parameter for Random Sources

Each of the random source blocks requires you to set a **Sample time** parameter in the block mask. If you configure the block to produce a sample-based signal, then this parameter is the time interval between successive updates of the signal. If you configure the block to produce a frame-based matrix signal, then the **Sample time** parameter is the time interval between successive rows of the frame-based matrix.

If you use a Simulink Probe block to query the period of a frame-based output from a random source block in the Comm Sources library, then note that the Probe block reports the period of the *entire frame*, not the period of *each sample* in a given channel of the frame. The equation below relates the quantities involved for a single-channel signal.

$$A \text{ seconds/frame} = (B \text{ seconds/sample}) * (S \text{ samples/frame})$$

where:

- A is the number shown in the Probe block after the Tf notation.
- B is the random source block's **Sample time** parameter.
- S is the random source block's **Samples per frame** parameter.

Seed Parameter for Random Sources

Each of the random source blocks requires you to set a seed in the block mask. This is the initial seed that the random number generator uses when forming its sequence of numbers. If you choose a constant seed, then the block produces the same noise sequence each time you start the simulation. The sequence will be different from that produced with a different constant seed. If you want the noise to be different each time you start the simulation, then you can use a varying seed such as `cputime`.

Signal Attribute Parameters for Random Sources

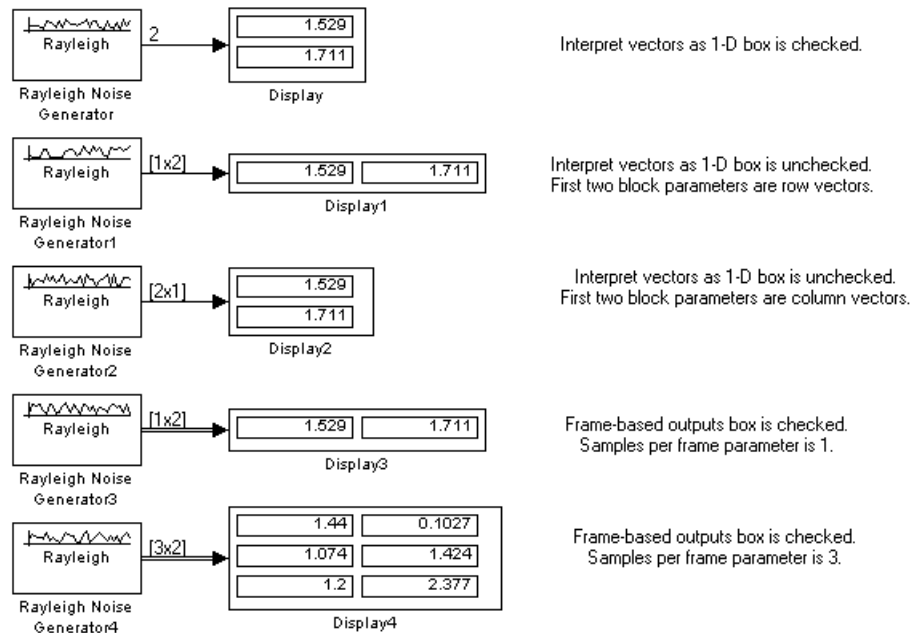
In most random source blocks, the output can be a frame-based matrix, a sample-based row or column vector, or a sample-based one-dimensional array.

The table below indicates how to set certain block parameters depending on the kind of signal you want to generate.

Signal Attributes	Parameter Settings
Sample-based, one-dimensional	<div><div><div><div><input type="checkbox"/> Frame-based outputs</div><div>Samples per frame:</div><div>1</div></div><div><input checked="" type="checkbox"/> Interpret vector parameters as 1-D</div></div></div>
Sample-based row vector	<div><div><div><div><input type="checkbox"/> Frame-based outputs</div><div>Samples per frame:</div><div>1</div></div><div><input type="checkbox"/> Interpret vector parameters as 1-D</div></div></div> <div>Also, any vector parameters in the block should be rows, not columns.</div>
Sample-based column vector	<div><div><div><div><input type="checkbox"/> Frame-based outputs</div><div>Samples per frame:</div><div>1</div></div><div><input type="checkbox"/> Interpret vector parameters as 1-D</div></div></div> <div>Also, any vector parameters in the block should be columns, not rows.</div>
Frame-based	<div><div><div><div><input checked="" type="checkbox"/> Frame-based outputs</div><div>Samples per frame:</div><div>number_of_rows</div></div><div><input type="checkbox"/> Interpret vector parameters as 1-D</div></div></div> <div>Also, set Samples per frame to the number of samples in each output frame, that is, the number of rows in the signal.</div>

The **Frame-based outputs** and **Interpret vector parameters as 1-D** check boxes are mutually exclusive, because frame-based signals and one-dimensional signals are mutually exclusive. The **Samples per frame** parameter field is active only if the **Frame-based outputs** check box is checked.

Example. The model in the figure below illustrates that one random source block can produce various kinds of signals. The annotations in the model indicate how each copy of the block is configured. Notice how each block's configuration affects the type of connector line (single or double) and the signal dimensions that appear above each connector line. In the case of the Rayleigh Noise Generator block, the first two block parameters (**Sigma** and **Initial seed**) determine the number of channels in the output; for analogous indicators in other random source blocks, see their individual reference entries.



Random Bits

The Bernoulli Random Binary Generator and Binary Vector Noise Generator blocks both generate random bits, but differ in the way that you specify the

distribution of 1s. As a result, the Bernoulli Random Binary Generator block is suitable for representing sources, while the Binary Vector Noise Generator block is more appropriate for modeling channel errors.

The Bernoulli Random Binary Generator block considers each element of the signal to be an independent Bernoulli random variable. Also, different elements need not be identically distributed.

The Binary Vector Noise Generator block constructs a random binary signal using a two-stage process. First, using information that you provide in the block mask, it determines how many 1s will appear. Then it determines where to place the required number of 1s, so that each possible arrangement has equal probability.

For example, if you set the **Binary vector length** parameter to 4, set the **Probabilities** parameter to 1, and uncheck the **Frame-based outputs** check box, then the block generates binary vectors of length 4, each of which contains exactly one 1. You might use these parameters to perturb a binary code that consists of four-bit codewords. Adding the random vector to your code vector (modulo 2) would introduce exactly one error into each codeword. Alternatively, to perturb each codeword by introducing one error with probability 0.4 and two errors with probability 0.6, set the **Probabilities** parameter to [0.4, 0.6] instead of 1.

Note that the **Probabilities** parameter of the Binary Vector Noise Generator block affects only the *number* of 1s in each vector, not their placement.

Random Integers

The Random-Integer Generator and Poisson Int Generator blocks both generate vectors containing random nonnegative integers. The Random-Integer Generator block uses a uniform distribution on a bounded range that you specify in the block mask. The Poisson Int Generator block uses a Poisson distribution to determine its output. In particular, the output can include any nonnegative integer.

Pseudorandom Symbols

The PN Sequence Generator block generates a sequence of pseudorandom symbols. Such a sequence can be used in a pseudorandom scrambler and descrambler, or in a direct-sequence spread-spectrum system.

Random Real Numbers

You can use one of several blocks to generate random real numbers, depending on what distribution you want to use. The choices are listed in the table below.

Distribution	Block
Gaussian	Gaussian Noise Generator
Rayleigh	Rayleigh Noise Generator
Rician	Rician Noise Generator
Uniform on a bounded interval	Uniform Noise Generator

The particular mask parameters depend on the block. See each block's individual entry in the reference section for details.

Nonrandom Signals

Blocks in the Comm Sources library can create nonrandom signals by reading from a file or by simulating a voltage-controlled oscillator (VCO):

- The Triggered Read from File block reads a record from a file whenever an input trigger signal has a rising edge. You can set up the block to read at every rising edge of the trigger, or every k th rising edge of the trigger for a positive number k .
- A voltage-controlled oscillator is one part of a phase-locked loop. The Voltage-Controlled Oscillator and Discrete-Time VCO blocks implement voltage-controlled oscillators. These blocks produce continuous-time and discrete-time output signals, respectively. Each block's output signal is sinusoidal, and changes its frequency in response to the amplitude variations of the input signal.

Communications Sinks

The Communications Blockset provides sinks and display devices that facilitate analysis of communication system performance. You can open the Comm Sinks library by double-clicking on its icon in the main Communications Blockset library (commlib), or by typing

```
commsink2
```

at the MATLAB prompt.

Sink Features of the Blockset

Blocks in this library can:

- Write to a file when trigger events occur
- Compute error statistics
- Plot an eye diagram
- Generate a scatter diagram

This section describes these capabilities. Other sinks are in Simulink's Sinks library and in the DSP Blockset's DSP Sinks library.

Writing to a File

The Triggered Write to File block writes data to a file whenever an input trigger signal has a rising edge. You can set up the block to write at every rising edge of the trigger, or every k th rising edge of the trigger for a positive number k . The data can have an ASCII, integer, or floating-point format. If the destination file already exists, then this block overwrites it. For more details, see the reference page for the Triggered Write to File block.

For untriggered writing of MAT files, use Simulink's To File block.

Error Statistics

The Error Rate Calculation block compares input data from a transmitter with input data from a receiver. It calculates these error statistics:

- The error rate
- The number of error events
- The total number of input events

The block reports these statistics either as final values in the workspace or as running statistics at an output port.

You can use this block either with binary inputs to compute the bit error rate, or with symbol inputs to compute the symbol error rate. You can use frame-based or sample-based data. Also, if you use frame-based data, then you can have the block consider certain samples and ignore others.

The example in the section “Examples of Convolutional Coding” on page 2-42 illustrates the use of the Error Rate Calculation block.

Eye Diagrams

An eye diagram is a simple and convenient tool for studying the effects of intersymbol interference and other channel impairments in digital transmission. When this blockset constructs an eye diagram, it plots the received signal against time on a fixed-interval axis. At the end of the fixed interval, it wraps around to the beginning of the time axis. Thus the diagram consists of many overlapping curves. One way to use an eye diagram is to look for the place where the “eye” is most widely opened, and use that point as the decision point when demapping a demodulated signal to recover a digital message.

The two blocks, Continuous-Time Eye and Scatter Diagrams and Discrete-Time Eye and Scatter Diagrams, both produce eye diagrams. One processes continuous-time signals and the other processes discrete-time signals. The blocks also differ in the way you determine the decision timing: the Continuous-Time Eye and Scatter Diagrams block draws a vertical line to indicate a decision every time a trigger signal has a rising edge, whereas the Discrete-Time Eye and Scatter Diagrams block draws a similar line periodically according to a mask parameter.

An example appears in “Example: Using Eye and Scatter Diagrams” on page 2-14.

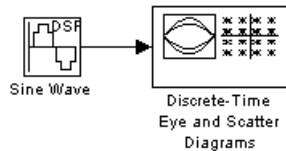
Scatter Diagrams

A scatter diagram of a signal plots the signal's value at a given decision point. In the best case, the decision point should be at the time when the eye of the signal's eye diagram is the most widely open.

The two blocks, Continuous-Time Eye and Scatter Diagrams and Discrete-Time Eye and Scatter Diagrams, both produce scatter diagrams. One processes continuous-time signals and the other processes discrete-time signals. They also differ in the way you determine the decision timing: the Continuous-Time Eye and Scatter Diagrams block plots a new point every time a trigger signal has a rising edge, whereas the Discrete-Time Eye and Scatter Diagrams block plots new points periodically according to a mask parameter.

Example: Using Eye and Scatter Diagrams

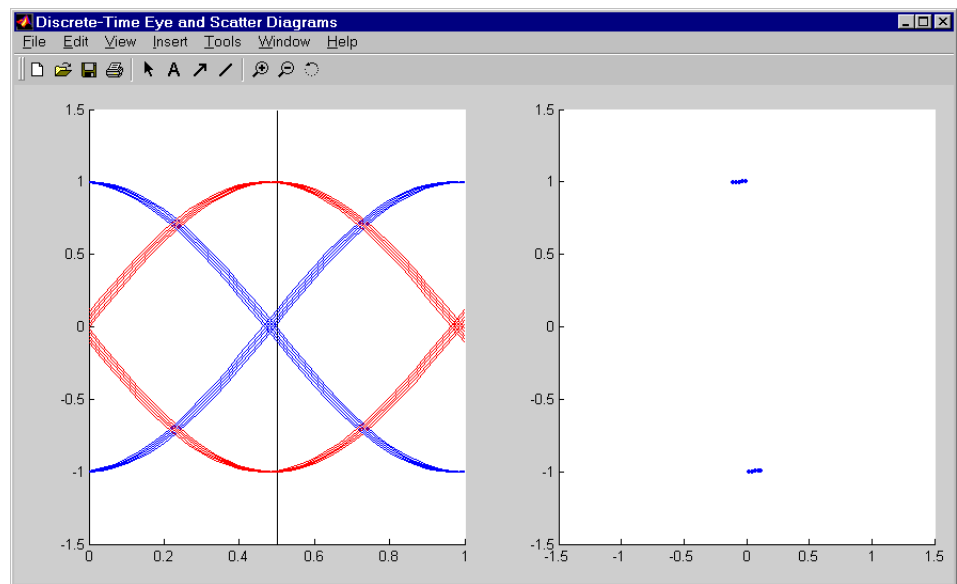
The model below creates an eye diagram and scatter diagram from a complex sinusoidal signal. Because the decision time interval is almost, but not exactly, an integer multiple of the period of the sinusoid, the diagram exhibits drift over time. More specifically, successive traces in the eye diagram and successive points in the scatter diagram are near each other but do not overlap.



To open the completed model, [click here](#) in the MATLAB Help browser. To build the model, gather and configure these blocks:

- Sine Wave, in the DSP Blockset DSP Sources library (*not* the Sine Wave block in the Simulink Sources library)
 - Set **Frequency** to . 502
 - Set **Output complexity** to **Complex**
- Discrete-Time Eye and Scatter Diagrams
 - Set **Diagram type** to **Eye and Scatter Diagrams**
 - Set **Sample time for plot update** to 1/30

Running the model produces the plots below. In the eye diagram, one set of traces represents the real part of the signal and the other set of traces represents the imaginary part of the signal.



Source Coding

Source coding, also known as *quantization* or *signal formatting*, is a way of processing data in order to reduce redundancy or prepare it for later processing. Analog-to-digital conversion and data compression are two categories of source coding.

Source coding divides into two basic procedures: *source encoding* and *source decoding*. Source encoding converts a source signal into a digital signal using a quantization method. The symbols in the resulting signal are nonnegative integers in some finite range. Source decoding recovers the original information from the source coded signal.

For background material on the subject of source coding, see the works listed in “Selected Bibliography for Source Coding” on page 2-26.

Source Coding Features of the Blockset

This blockset supports scalar quantization, predictive quantization, companders, and differential coding. It does not support vector quantization. You can open the Source Coding library by double-clicking on its icon in the main Communications Blockset library (commlib), or by typing

```
commsrccod2
```

at the MATLAB prompt.

Blocks in the Source Coding library can:

- Use a partition and codebook to quantize a signal
- Implement differential pulse code modulation (DPCM)
- Compand a signal using a μ -law or A-law compressor or expander
- Encode or decode a signal using differential coding

Supporting functions in the Communications Toolbox also allow you to optimize source coding parameters for a set of training data. See the sections “Optimizing Quantization Parameters” and “Optimizing DPCM Parameters” in the *Communications Toolbox User’s Guide* for more information about such capabilities.

Representing Quantization Parameters

Scalar quantization is a process that maps all inputs within a specified range to a common value. It maps inputs in a different range of values to a different common value. In effect, scalar quantization digitizes an analog signal. Two parameters determine a quantization: a partition and a codebook. This section describes how blocks represent these parameters.

Partitions

A quantization partition defines several contiguous, nonoverlapping ranges of values within the set of real numbers. To specify a partition as a parameter, list the distinct endpoints of the different ranges in a vector.

For example, if the partition separates the real number line into the sets

- $\{x: x \leq 0\}$
- $\{x: 0 < x \leq 1\}$
- $\{x: 1 < x \leq 3\}$
- $\{x: 3 < x\}$

then you can represent the partition as the three-element vector

$[0, 1, 3]$

Notice that the length of the partition vector is one less than the number of partition intervals.

Codebooks

A codebook tells the quantizer which common value to assign to inputs that fall into each range of the partition. Represent a codebook as a vector whose length is the same as the number of partition intervals. For example, the vector

$[-1, 0, 5, 2, 3]$

is one possible codebook for the partition $[0, 1, 3]$.

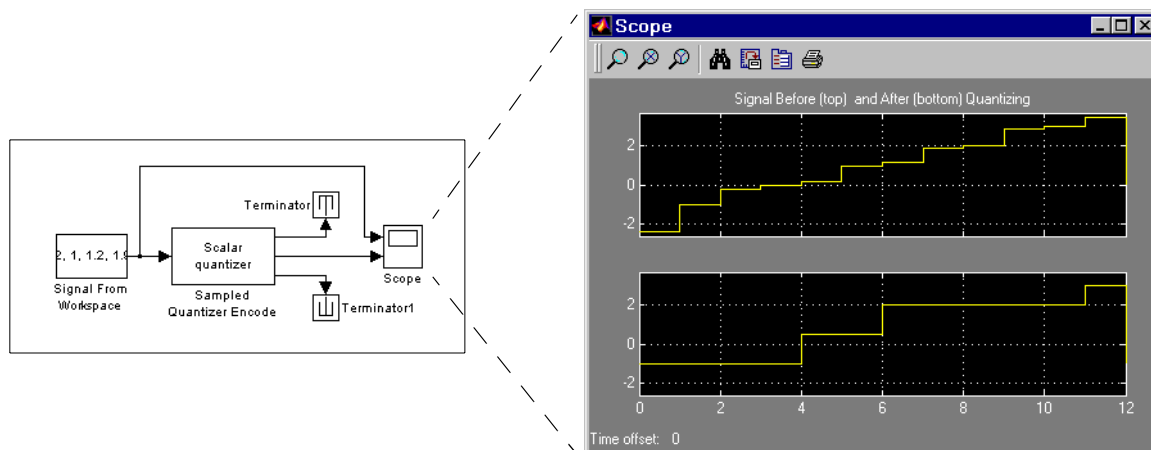
Quantizing a Signal

This section shows how the Sampled Quantizer Encode, Enabled Quantizer Encode, and Quantizer Decode blocks use the partition and codebook parameters. (The Enabled Quantizer Encode block does not appear in an example, but its behavior is similar to that of the Sampled Quantizer Encode

block.) The examples here are analogous to “Scalar Quantization Example 1” and “Scalar Quantization Example 2” in the *Communications Toolbox User’s Guide*.

Scalar Quantization Example 1

The figure below shows how the Sampled Quantizer Encode block uses the partition and codebook as defined above to map a real vector to a new vector whose entries are either -1, 0.5, 2, or 3. In the Scope window, the bottom signal is the quantization of the (original) top signal.



To open the completed model, click [here](#) in the MATLAB Help browser. To build the model, gather and configure these blocks:

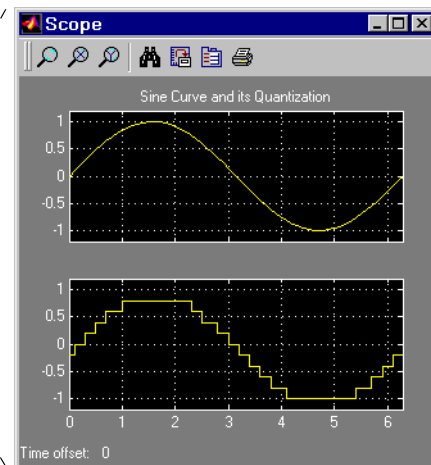
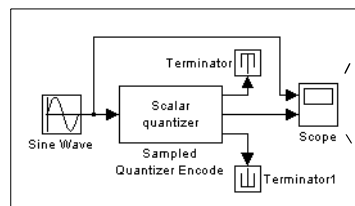
- Signal From Workspace, in the DSP Blockset DSP Sources library
 - Set **Signal** to [- 2. 4, - 1, - . 2, 0, . 2, 1, 1. 2, 1. 9, 2, 2. 9, 3, 3. 5]'
- Sampled Quantizer Encode
 - Set **Quantization partition** to [0, 1, 3]
 - Set **Quantization codebook** to [- 1, 0. 5, 2, 3]
 - Set **Input signal vector length** to 1
 - Set **Sample time** to 1
- Terminator, in the Simulink Signals & Systems library

- Scope, in the Simulink Sinks library
 - After double-clicking on the block to open it, click on the **Properties** icon and set **Number of axes** to 2.

Connect the blocks as shown in the figure. Also, from the model window's **Simulation menu**, choose **Simulation parameters**; then in the **Simulation Parameters** dialog box, set **Stop time** to 12. Running the model produces a scope image similar to the one in the figure. (To make the axis ranges and title exactly match those in the figure, right-click on each plot area in the scope and select **Axes properties**.)

Scalar Quantization Example 2

This example, shown in the figure below, illustrates the nature of scalar quantization more clearly. It quantizes a sampled sine wave and plots the original (top) and quantized (bottom) signals. The plot contrasts the smooth sine curve with the polygonal curve of the quantized signal. The vertical coordinate of each flat part of the polygonal curve is a value in the **Quantization codebook** vector.



To open the completed model, click [here](#) in the MATLAB Help browser. To build the model, gather and configure these blocks:

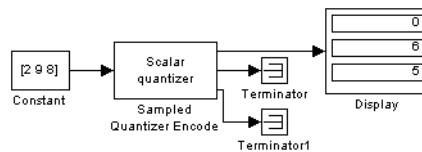
- Sine Wave, in the Simulink Sources library (*not* the Sine Wave block in the DSP Blockset DSP Sources library)

- Sampled Quantizer Encode
 - Set **Quantization partition** to [- 1: . 2: 1]
 - Set **Quantization codebook** to [- 1. 2: . 2: 1]
 - Set **Input signal vector length** to 1
- Terminator, in the Simulink Signals & Systems library
- Scope, in the Simulink Sinks library
 - After double-clicking on the block to open it, click on the **Properties** icon and set **Number of axes** to 2.

Connect the blocks as shown in the figure. Also, from the model window's **Simulation menu**, choose **Simulation parameters**; then in the **Simulation Parameters** dialog box, set **Stop time** to 2π . Running the model produces the scope image as shown in the figure. (To make the axis ranges and title exactly match those in the figure, right-click on each plot area in the scope and select **Axes properties**.)

Determining Which Interval Each Input Is in

The Sampled Quantizer Encode block also returns a signal, at the first output port, that tells which interval each input is in. For example, the model below shows that the input entries lie within the intervals labeled 0, 6, and 5, respectively. Here, the 0th interval consists of real numbers less than or equal to 3; the 6th interval consists of real numbers greater than 8 but less than or equal to 9; and the 5th interval consists of real numbers greater than 7 but less than or equal to 8.



To open the completed model, click [here](#) in the MATLAB Help browser. To build the model, gather and configure these blocks:

- Constant, in the Simulink Sources library
 - Set **Constant value** to [2, 9, 8]
- Sampled Quantizer Encode
 - Set **Quantization partition** to [3, 4, 5, 6, 7, 8, 9]

- Set **Quantization codebook** to any vector whose length exceeds the length of **Quantization Partition** by one
- Set **Input signal vector length** to 3
- Terminator, in the Simulink Signals & Systems library
- Display, in the Simulink Sinks library
 - Drag the bottom edge of the icon to make the display big enough for three entries

Connect the blocks as shown above. Running the model produces the display numbers as shown in the figure.

You can continue this example by branching the first output of the Sampled Quantizer Encode block, connecting one branch to the input port of the Quantizer Decode block, and connecting the output of the Quantizer Decode block to another Display block. If the two source coding blocks' **Quantization codebook** parameters match, then the output of the Quantizer Decode block will be the same as the second output of the Sampled Quantizer Encode block. Thus the Quantizer Decode block partially duplicates the functionality of the Sampled Quantizer Encode block, but requires different input data and fewer parameters.

Implementing Differential Pulse Code Modulation

The quantization in the section “Quantizing a Signal” on page 2-17 requires no *a priori* knowledge about the transmitted signal. In practice, you can often make educated guesses about the present signal based on past signal transmissions. Using such educated guesses to help quantize a signal is known as *predictive quantization*. The most common predictive quantization method is differential pulse code modulation (DPCM). The DPCM Encoder and DPCM Decoder blocks can help you implement a DPCM predictive quantizer.

DPCM Terminology

To determine an encoder for such a quantizer, you must supply not only a partition and codebook as described in “Representing Quantization Parameters” on page 2-17, but also a *predictor*. The predictor is a function that the DPCM encoder uses to produce the educated guess at each step. Instead of quantizing x itself, the encoder quantizes the *predictive error*, which is the difference between the educated guess and the actual value. The special case when the numerator is linear and the denominator is 1 is called *delta modulation*.

For more information about how DPCM works, see [1] in “Selected Bibliography for Source Coding” on page 2-26, or look underneath the masks of the DPCM Encoder and DPCM Decoder blocks.

Representing Predictors

This blockset implements predictors using an IIR filter. Just as you can specify a filter using a rational function of z^{-1} , you specify the predictor by giving its numerator and denominator. In block masks, the numerator and denominator are vectors that list the coefficients in order of ascending powers of z^{-1} .

The numerator’s constant term must be zero. This makes sense conceptually because the filter’s output is meant to *predict* the present signal without actually knowing its value.

In most applications, the denominator is the constant function 1.

Coded and Decoded Signals

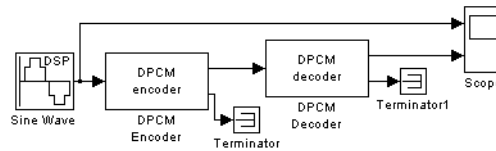
If you encode a given signal using DPCM, then two resulting signals are the quantization index and the quantization-encoded signal. These correspond exactly to the two outputs of an ordinary quantization encoder. In both instances, the quantization index tells which partition interval a signal lies in, and the quantization-encoded signal tells which codebook values correspond to those partition intervals.

To use the DPCM Decoder block to recover a message that has been through the DPCM Encoder block, connect the quantization index signal, *not* the quantization-encoded signal, to the input port of the DPCM Decoder block.

The DPCM Decoder block outputs two signals. The first output is the attempted recovery of the message that first entered the DPCM encoder (assuming the encoder and decoder have matching parameters). The second output comes directly from the underlying quantization decoder. It represents the quantized predictive error, not the recovered message itself.

Example: Using DPCM Encoding and Decoding

A simple special case of DPCM quantizes the difference between the signal’s current value and its value at the previous step. Thus the predicted value equals the actual value at the previous step. The model below implements this scheme. It encodes a sine wave, decodes it, and plots both the original and decoded signals.

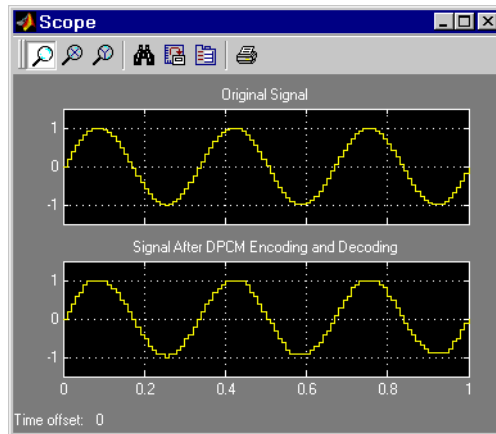


To open the completed model, click [here](#) in the MATLAB Help browser. To build the model, gather and configure these blocks:

- Sine Wave, in the DSP Blockset DSP Sources library (*not* the Sine Wave block in the Simulink Sources library)
 - Set **Frequency** to 3
 - Set **Sample time** to .01
- DPCM Encoder
 - Set **Predictor numerator** to [0, 1]
 - Set **Quantization partition** to [-10:9]/10
 - Set **Quantization codebook** to [-10:10]/10
 - Set **Sample time** to .01
- DPCM Decoder
 - Set **Predictor numerator**, **Quantization codebook**, and **Sample time** to the values given for the DPCM Encoder block
- Terminator, in the Simulink Signals & Systems library
- Scope, in the Simulink Sinks library
 - After double-clicking on the block to open it, click on the **Properties** icon and set **Number of axes** to 2.

Connect the blocks as shown in the figure. Also, from the model window's **Simulation menu**, choose **Simulation parameters**; then in the **Simulation Parameters** dialog box, set **Stop time** to 1.

Running the model produces scope images similar to those below. (To make the axis ranges and titles exactly match those below, right-click on each plot area in the scope and select **Axes properties**.)



Companding a Signal

In certain applications, such as speech processing, it is common to use a logarithm computation, called a *compressor*, before quantizing. The inverse operation of a compressor is called an *expander*. The combination of a compressor and expander is called a *compander*.

This blockset supports two kinds of companders: μ -law and A-law companders. The reference pages for the A-Law Compressor, A-Law Expander, Mu-Law Compressor, and Mu-Law Expander blocks list the relevant expander and compressor laws.

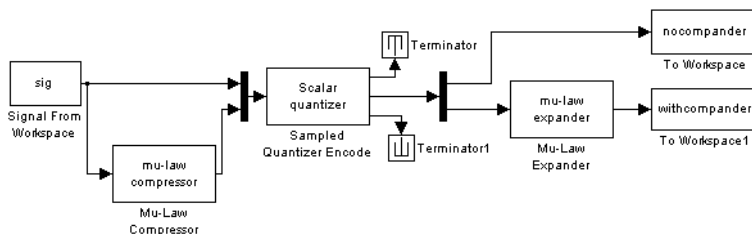
Example: Using a μ -Law Compander

This example quantizes an exponential signal in two ways and compares the resulting mean square distortions. To create the signal in the MATLAB workspace, execute these commands:

```
sig = -4: .1: 4;
sig = exp(sig'); % Exponential signal to quantize
```

Now, the model in the figure below performs two computations. One computation uses the Sampled Quantizer Encode block with a partition consisting of length-one intervals. The second computation uses the Mu-Law Compressor block to implement a μ -law compressor, the Sampled Quantizer

Encode block to quantize the compressed data and, finally, the Mu-Law Expander block to expand the quantized data.



To open the completed model, click here in the MATLAB Help browser. To build the model, gather and configure these blocks:

- Signal From Workspace, in the DSP Blockset DSP Sources library
 - Set **Signal** to `si g`
- Mu-Law Compressor
 - Set **Peak signal magnitude** to `max(si g)`
- Mux, in the Simulink Signals & Systems library
- Sampled Quantizer Encode, in the Source Coding library
 - Set **Quantization partition** to `0: floor(max(si g))`
 - Set **Quantization codebook** to `0: ceil(max(si g))`
 - Set **Sample time** to 1
- Terminator, in the Simulink Signals & Systems library
- Demux, in the Simulink Signals & Systems library
- Mu-Law Expander
 - Set **Peak signal magnitude** to `ceil(max(si g))`
- Two copies of To Workspace, in the Simulink Sinks library
 - Set **Variable name** to `nocompander` and `withcompander`, respectively, in the two copies of this block
 - Set **Save format** to **Array** in each of the two copies of this block

Connect the blocks as shown above. Also, from the model window's **Simulation menu**, choose **Simulation parameters**; then in the **Simulation Parameters** dialog box, set **Stop time** to 80. Run the model, then execute these commands:

```
di stor = sum((nocompander-si g).^2)/length(si g);  
di stor2 = sum((withcompander-si g).^2)/length(si g);  
[di stor di stor2]
```

```
ans =
```

```
0.5348    0.0397
```

This output shows that the distortion is smaller for the second scheme. This is because equal-length intervals are well suited to the logarithm of the data but not as well suited to the data itself.

Selected Bibliography for Source Coding

[1] Kondo, A. M. *Digital Speech*. Chichester, England: John Wiley & Sons, 1994.

[2] Sklar, Bernard. *Digital Communications: Fundamentals and Applications*. Englewood Cliffs, N.J.: Prentice-Hall, 1988.

Block Coding

Error-control coding techniques detect and possibly correct errors that occur when messages are transmitted in a digital communication system. To accomplish this, the encoder transmits not only the information symbols but also extra redundant symbols. The decoder interprets what it receives, using the redundant symbols to detect and possibly correct whatever errors occurred during transmission. You might use error-control coding if your transmission channel is very noisy or if your data is very sensitive to noise. Depending on the nature of the data or noise, you might choose a specific type of error-control coding.

Block coding is a special case of error-control coding. Block coding techniques maps a fixed number of message symbols to a fixed number of code symbols. A block coder treats each block of data independently and is a memoryless device.

Organization of This Section

These topics provide background information:

- “Accessing Block Coding Blocks” on page 2-27
- “Block Coding Features of the Blockset” on page 2-28
- “Communications Toolbox Support Functions” on page 2-29
- “Channel Coding Terminology” on page 2-29

These topics describe how to simulate linear block coding:

- “Data Formats for Block Coding” on page 2-29
- “Using Block Encoders and Decoders Within a Model” on page 2-32
- “Examples of Block Coding” on page 2-32
- “Notes on Specific Block Coding Techniques” on page 2-35

For background material on the subject of block coding, see the works listed in “Selected Bibliography for Block Coding” on page 2-39.

Accessing Block Coding Blocks

You can open the Channel Coding library by double-clicking on its icon in the main Communications Blockset library (comm1 i b), or by typing

```
commfecod2
```

at the MATLAB prompt.

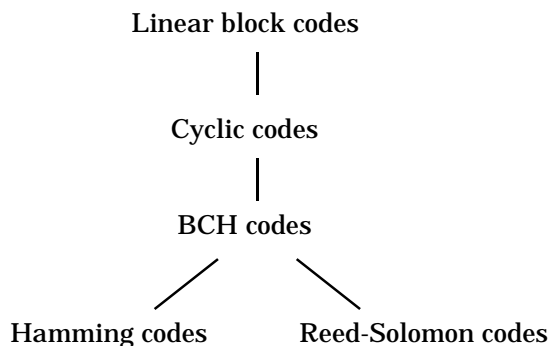
Then you can open the Block sublibrary by double-clicking on its icon in the Channel Coding library, or by typing

```
commbl kcod2
```

at the MATLAB prompt.

Block Coding Features of the Blockset

The class of block coding techniques includes categories shown in the diagram below.



The Communications Blockset supports general linear block codes. It also includes blocks that process cyclic, BCH, Hamming, and Reed-Solomon codes (which are all special kinds of linear block codes). Blocks in the blockset can encode or decode a message using one of the techniques mentioned above. The Reed-Solomon and BCH decoders indicate how many errors they detected while decoding. The Reed-Solomon coding blocks also let you decide whether to use symbols or bits as your data.

Note The blocks in this blockset are designed for error-control codes that use an alphabet having 2 or 2^m symbols.

Communications Toolbox Support Functions

Functions in the Communications Toolbox can support the Communications Blockset simulation blocks by:

- Determining characteristics of a technique, such as error-correction capability or possible message lengths
- Performing lower-level computations associated with a technique, such as:
 - Computing a truth table
 - Computing a generator or parity-check matrix
 - Converting between generator and parity-check matrices
 - Computing a generator polynomial

For more information about error-control coding capabilities of the Communications Toolbox, see the section “Block Coding” in the *Communications Toolbox User's Guide*.

Channel Coding Terminology

Throughout this section, the information to be encoded consists of *message* symbols and the code that is produced consists of *codewords*.

Each block of K message symbols is encoded into a codeword that consists of N message symbols. K is called the message length, N is called the codeword length, and the code is called an $[N,K]$ code.

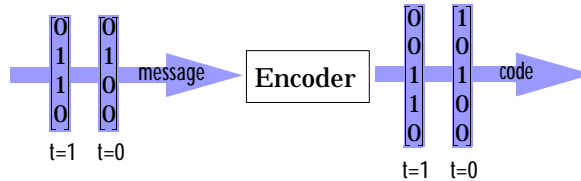
Data Formats for Block Coding

Each message or codeword is an ordered grouping of symbols. Each block in the Block Coding sublibrary processes one word in each time step, as described in “Binary Format (All Coding Methods)” below. Reed-Solomon coding blocks also let you choose between binary and integer data, as described in “Integer Format (Reed-Solomon only)” on page 2-31.

Binary Format (All Coding Methods)

You can structure messages and codewords as binary *vector* signals, where each vector represents a message word or a codeword. At a given time, the encoder receives an entire message word, encodes it, and outputs the entire codeword. The message and code signals share the same sample time.

The figure below illustrates this situation. In this example, the encoder receives a four-bit message and produces a five-bit codeword at time 0. It repeats this process with a new message at time 1.



For all coding techniques *except* Reed-Solomon, the message vector must have length K and the corresponding code vector has length N . For Reed-Solomon codes, the message vector must have length $M \cdot K$ and the corresponding code vector has length $M \cdot N$. Here M is an integer greater than or equal to 3 that satisfies $N = 2^M - 1$.

If the input to a block coding block is a frame-based vector, then it must be a column vector instead of a row vector.

To produce sample-based messages in the binary format, you can configure the Bernoulli Random Binary Generator block so that its **Probability of a zero** parameter is a vector whose length is that of the signal you want to create. To produce frame-based messages in the binary format, you can configure the same block so that its **Probability of a zero** parameter is a scalar and its **Samples per frame** parameter is the length of the signal you want to create.

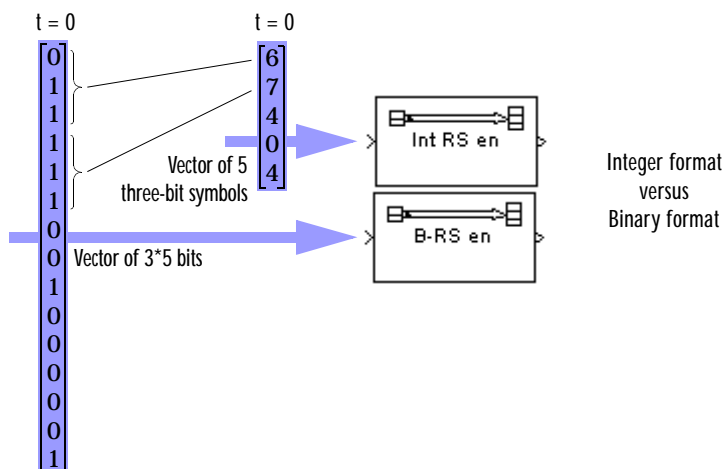
Using Serial Signals. If you prefer to structure messages and codewords as scalar signals, where several samples jointly form a message word or codeword, then you can use the Buffer and Unbuffer blocks in the DSP Blockset. Be aware that buffering involves latency and multirate processing. See the reference page for the Buffer block for more details. If your model computes error rates, then the initial delay in the coding-buffering combination influences the **Receive delay** parameter in the Error Rate Calculation block. If you are unsure about the sample times of signals in your model, then selecting **Sample time colors** from the model's **Format** menu, or attaching Probe blocks (from the Simulink Signals & Systems library) to connector lines might help.

Integer Format (Reed-Solomon only)

A message word for an $[N, K]$ Reed-Solomon code consists of $M \cdot K$ bits, which you can interpret as K symbols between 0 and 2^M . Here $N = 2^M - 1$ and M is an integer greater than or equal to 3. The integer format for Reed-Solomon codes lets you structure messages and codewords as *integer* signals instead of binary signals. (If the input is a frame-based vector, then it must be a column vector instead of a row vector.)

Note In this context, Simulink expects the *first* bit to be the least significant bit in the symbol. “First” means the smallest index in a vector or the smallest time for a series of scalars.

The figure below illustrates the equivalence between binary and integer signals for a Reed-Solomon encoder. The case for the decoder would be similar.



To produce sample-based messages in the integer format, you can configure the Random-Integer Generator block so that **M-ary number** and **Initial seed** parameters are vectors of the desired length and all entries of the **M-ary number** vector are 2^M . To produce frame-based messages in the integer format, you can configure the same block so that its **M-ary number** and **Initial seed** parameters are scalars and its **Samples per frame** parameter is the length of the signal you want to create.

Using Block Encoders and Decoders Within a Model

Once you have configured the coding blocks, a few tips will help you place them correctly within your model:

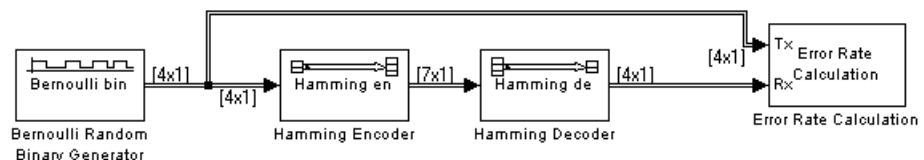
- If a block has multiple outputs, then the first one is always the stream of coding data.
The Reed-Solomon and BCH blocks have an error counter as a second output.
- Be sure the signal sizes are appropriate for the mask parameters. For example, if you use the Binary Cyclic Encoder block and set **Message length K** to 4, then the input signal must be a vector of length 4.
If you are unsure about the size of signals in your model, then selecting **Signal dimensions** from the model's **Format** menu may help.

Examples of Block Coding

This section presents two example models. The first example processes a Hamming code using the binary format and the second example processes a Reed-Solomon code using the integer format.

Example: Hamming Code in Binary Format

This example shows very simply how to use an encoder and decoder. It illustrates the appropriate vector lengths of the code and message signals for the coding blocks. Also, because the Error Rate Calculation block accepts only scalars or frame-based column vectors as the transmitted and received signals, this example uses frame-based column vectors throughout. (It thus avoids having to change signal attributes using a block such as Convert 1-D to 2-D.)



To open the completed model, click [here](#) in the MATLAB Help browser. To build the model, gather and configure these blocks:

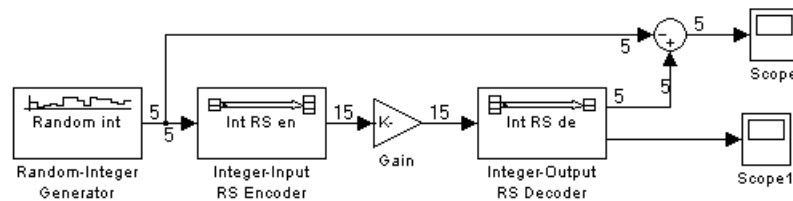
- Bernoulli Random Binary Generator, in the Comm Sources library
 - Set **Probability of a zero** to . 5

- Set **Initial seed** to any positive integer scalar
- Check the **Frame-based outputs** check box
- Set **Samples per frame** to 4
- Hamming Encoder, with default parameter values
- Hamming Decoder, with default parameter values
- Error Rate Calculation, in the Comm Sinks library, with default parameter values

Connect the blocks as in the figure above. Also, use the **Signal dimensions** feature from the model window's **Format** menu. After possibly updating the diagram if necessary (**Update diagram** from the **Edit** menu), the connector lines show relevant signal attributes. The connector lines are double lines to indicate frame-based signals, and the annotations next to the lines show that the signals are column vectors of appropriate sizes.

Example: Reed-Solomon Code in Integer Format

This example uses a Reed-Solomon code in integer format. It illustrates the appropriate vector lengths of the code and message signals for the coding blocks. It also exhibits error correction, using a very simplistic way of introducing errors into each codeword.



To open the completed model, click [here](#) in the MATLAB Help browser. To build the model, gather and configure these blocks:

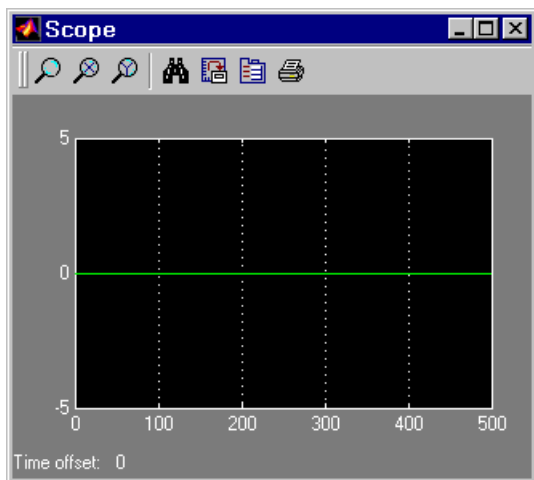
- Random-Integer Generator, in the Comm Sources library
 - Set **M-ary number** to 15
 - Set **Initial seed** to any vector containing 5 positive integers
 - Check the **Interpret vector parameters as 1-D** check box

- Integer-Input RS Encoder
 - Set **Codeword length N** to 15
- Gain, in the Simulink Math library
 - Set **Gain** to [0 0 0 0 0, ones(1, 10)]
- Integer-Output RS Decoder
 - Set **Codeword length N** to 15
- Scope, in the Simulink Sinks library. Get two copies of this block.
- Sum, in the Simulink Math library
 - Set **List of signs** to | - +

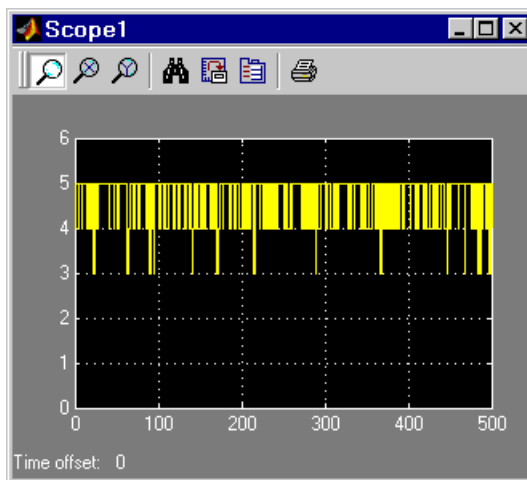
Connect the blocks as in the figure above. Also, from the model window's **Simulation menu**, choose **Simulation parameters**; then in the **Simulation Parameters** dialog box, set **Stop time** to 500.

The vector length numbers appear on the connecting lines only if you select **Signal dimensions** from the model's **Format** menu. Notice that the encoder accepts a vector of length 5 (which is K in this case) and produces a vector of length 15 (which is N in this case). The decoder does the opposite. Also, the **Initial seed** parameter in the Random-Integer Generator block is a vector of length 5 because it must generate a message word of length 5.

Running the model produces the scope images below. Your plot of the error counts might differ somewhat, depending on your **Initial seed** value in the Random-Integer Generator block. (To make the axis range exactly match that of the left scope in the figure, right-click on the plot area in the scope and select **Axes properties**.)



Difference Between Original Message and Recovered Message



Number of Errors Before Correction

The plot on the right is the number of errors that the decoder detected while trying to recover the message. Often the number is five because the Gain block replaces the first five symbols in each codeword with zeros. However, the number of errors is less than five whenever a correct codeword contains one or more zeroes in the first five places.

The plot on the left is the difference between the original message and the recovered message; since the decoder was able to correct all errors that occurred, each of the five data streams in the plot is zero.

Notes on Specific Block Coding Techniques

Although the Block Coding sublibrary is somewhat uniform in its look and feel, the various coding techniques are not identical. This section describes special options and restrictions that apply to parameters and signals for the coding technique categories in this sublibrary. You should read the part that applies to the coding technique you want to use: generic linear block code, cyclic code, Hamming code, BCH code, or Reed-Solomon code.

Generic Linear Block Codes

Encoding a message using a generic linear block code requires a generator matrix. Decoding the code requires the generator matrix and possibly a truth table. In order to use the Binary Linear Encoder and Binary Linear Decoder blocks, you must understand the **Generator matrix** and **Error-correction truth table** parameters.

Generator Matrix. The process of encoding a message into an $[N, K]$ linear block code is determined by a K -by- N generator matrix G . Specifically, a 1 -by- K message vector v is encoded into the 1 -by- N codeword vector vG . If G has the form $[I_k, P]$ or $[P, I_k]$, where P is some K -by- $(N-K)$ matrix and I_k is the K -by- K identity matrix, then G is said to be in *standard form*. (Some authors, such as Clark and Cain [1], use the first standard form, while others, such as Lin and Costello [2], use the second.) The linear block coding blocks in this blockset require the **Generator matrix** mask parameter to be in standard form.

Decoding Table. A decoding table tells a decoder how to correct errors that might have corrupted the code during transmission. Hamming codes can correct any single-symbol error in any codeword. Other codes can correct, or partially correct, errors that corrupt more than one symbol in a given codeword.

The Binary Linear Decoder block allows you to specify a decoding table in the **Error-correction truth table** parameter. Represent a decoding table as a matrix with N columns and 2^{N-K} rows. Each row gives a correction vector for one received codeword vector.

If you do not want to specify a decoding table explicitly, then set that parameter to 0. This causes the block to compute a decoding table using the `syndtbl` function in the Communications Toolbox.

Cyclic Codes

For cyclic codes, the codeword length N must have the form $2^M - 1$, where M is an integer greater than or equal to 3.

Generator Polynomials. Cyclic codes have special algebraic properties that allow a polynomial to determine the coding process completely. This so-called generator polynomial is a degree- $(N-K)$ divisor of the polynomial $x^N - 1$. Van Lint [4] explains how a generator polynomial determines a cyclic code.

The Binary Cyclic Encoder and Binary Cyclic Decoder blocks allow you to specify a generator polynomial as the second mask parameter, instead of

specifying K there. The blocks represent a generator polynomial using a vector that lists the polynomial's coefficients in order of *ascending* powers of the variable. You can find generator polynomials for cyclic codes using the `cyclpoly` function in the Communications Toolbox.

If you do not want to specify a generator polynomial, then set the second mask parameter to the value of K .

Hamming Codes

For Hamming codes, the codeword length N must have the form $2^M - 1$, where M is an integer greater than or equal to 3. The message length K must equal $N - M$.

Primitive Polynomials. Hamming codes rely on algebraic fields that have 2^M elements (or, more generally, p^M elements for a prime number p). Elements of such fields are named *relative to* a distinguished element of the field that is called a primitive element. The minimal polynomial of a primitive element is called a primitive polynomial. The Hamming Encoder and Hamming Decoder blocks allow you to specify a primitive polynomial for the finite field that they use for computations. If you want to specify this polynomial, then do so in the second mask parameter field. The blocks represent a primitive polynomial using a vector that lists the polynomial's coefficients in order of *ascending* powers of the variable. You can find generator polynomials for Galois fields using the `gfprimd` function in the Communications Toolbox.

If you do not want to specify a primitive polynomial, then set the second mask parameter to the value of K .

BCH Codes

For BCH codes, the codeword length N must have the form $2^M - 1$, where M is an integer greater than or equal to 3. The message length K can have only particular values. To see which values of K are valid for a given N , use the `bchpoly` function in the Communications Toolbox. For example, in the output below, the second column lists all possible message lengths that correspond to a codeword length of 31. The third column lists the corresponding error-correction capabilities.

```
params = bchpoly(31)
```

params =

31	26	1
31	21	2
31	16	3
31	11	5
31	6	7

No known analytic formula describes the relationship among the codeword length, message length, and error-correction capability for BCH codes.

Error Information. The BCH Decoder block allows you to state the error-correction capability of the code, as the **Error-correction capability T** parameter. Providing the value here speeds the computation. If you do not know the code's error-correction capability, then setting this parameter to zero causes the block to calculate the error-correction capability when initializing. You can find out the error-correction capability using the `bchpoly` function in the Communications Toolbox.

The BCH Decoder block also returns error-related information during the simulation. The second output signal indicates the number of errors that the block detected in the input codeword. A negative integer in the second output indicates that the block detected more errors than it could correct using the coding scheme.

Reed-Solomon Codes

Reed-Solomon codes are useful for correcting errors that occur in bursts. The codeword length N of a Reed-Solomon code must have the form $2^M - 1$, where M is an integer greater than or equal to 3. The error correction capability of a Reed-Solomon code is $\text{floor}((N - K) / 2)$. Since N is an odd number, the coding is more efficient when the message length K is also odd.

Effect of Nonbinary Symbols. One difference between Reed-Solomon codes and the other codes supported in this blockset is that Reed-Solomon codes process symbols in $\text{GF}(2^M)$ instead of $\text{GF}(2)$. Each such symbol is specified by M bits. The nonbinary nature of the Reed-Solomon code symbols causes the Reed-Solomon blocks to differ from other coding blocks in these ways:

- You can use the integer format, via the Integer-Input RS Encoder and Integer-Output RS Decoder blocks.

- The binary format forms a message from $M \cdot K$ (not K) samples and a codeword from $M \cdot N$ (not N) samples.
- The binary format expects the vector lengths to be $M \cdot K$ (not K) for messages and $M \cdot N$ (not N) for codewords.

Error Information. The Reed-Solomon decoding blocks (Binary-Output RS Decoder and Integer-Output RS Decoder) return error-related information during the simulation. The second output signal indicates the number of errors that the block detected in the input codeword. A negative integer in the second output indicates that the block detected more errors than it could correct using the coding scheme.

Selected Bibliography for Block Coding

- [1] Clark, George C. Jr. and J. Bibb Cain. *Error-Correction Coding for Digital Communications*. New York: Plenum Press, 1981.
- [2] Lin, Shu and Daniel J. Costello, Jr. *Error Control Coding: Fundamentals and Applications*. Englewood Cliffs, N.J.: Prentice-Hall, 1983.
- [3] Peterson, W. Wesley and E. J. Weldon, Jr. *Error-correcting Codes*, 2nd ed. Cambridge, Mass.: MIT Press, 1972.
- [4] van Lint, J. H. *Introduction to Coding Theory*. New York: Springer-Verlag, 1982.

Convolutional Coding

Convolutional coding is a special case of error-control coding. Unlike a block coder, a convolutional coder is not a memoryless device. Even though a convolutional coder accepts a fixed number of message symbols and produces a fixed number of code symbols, its computations depend not only on the current set of input symbols but on some of the previous input symbols.

Organization of This Section

- Describes how to access the Convolutional sublibrary of Channel Coding
- Summarizes the software's capabilities
- Discusses parameters for convolutional coding (polynomial description and trellis description)
- Gives examples of convolutional coding

For background material on the subject of convolutional coding, see the works listed in "Selected Bibliography for Convolutional Coding" on page 2-45.

Accessing Convolutional Coding Blocks

You can open the Channel Coding library by double-clicking on its icon in the main Communications Blockset library (`comm1 i b`), or by typing

```
commfeccod2
```

at the MATLAB prompt.

Then you can open the Convolutional sublibrary by double-clicking on its icon in the Channel Coding library, or by typing

```
commconvcod2
```

at the MATLAB prompt.

Convolutional Coding Features of the Blockset

The Communications Blockset supports feedforward or feedback binary convolutional codes that can be described by a trellis structure or a set of generator polynomials. It uses the Viterbi algorithm to implement hard-decision and soft-decision decoding.

The blockset also includes an *a posteriori* probability decoder, which can be useful for processing turbo codes.

Parameters for Convolutional Coding

To process convolutional codes (including turbo codes), use the Convolutional Encoder, Viterbi Decoder, and/or APP Decoder blocks in the Convolutional sublibrary. If a mask parameter is required in both the encoder and the decoder, then use the same value in both blocks.

The blocks in the Convolutional sublibrary assume that you use one of two different representations of a convolutional encoder:

- If you design your encoder using a diagram with shift registers and modulo-2 adders, then you can compute the code generator polynomial matrix and subsequently use the `poly2trellis` function (in the Communications Toolbox) to generate the corresponding trellis structure mask parameter automatically. For an example, see “Example: A Rate 2/3 Feedforward Convolutional Encoder” on page 2-42.
- If you design your encoder using a trellis diagram, then you can construct the trellis structure in MATLAB and use it as the mask parameter.

Details about these representations are in the sections, “Polynomial Description of a Convolutional Encoder” and “Trellis Description of a Convolutional Encoder,” in the *Communications Toolbox User's Guide*.

Using the Polynomial Description in Blocks

To use the polynomial description with the Convolutional Encoder, Viterbi Decoder, or APP Decoder blocks, you can use the utility function `poly2trellis`, from the Communications Toolbox. This function accepts a polynomial description and converts it into a trellis description. For example, the command below computes the trellis description of an encoder whose constraint length is five and whose generator polynomials are 35 and 31.

```
trellis = poly2trellis(5, [35 31]);
```

To use this encoder with one of the convolutional coding blocks, simply place a `poly2trellis` command such as

```
poly2trellis(5, [35 31]);
```

in the **Trellis structure** parameter field.

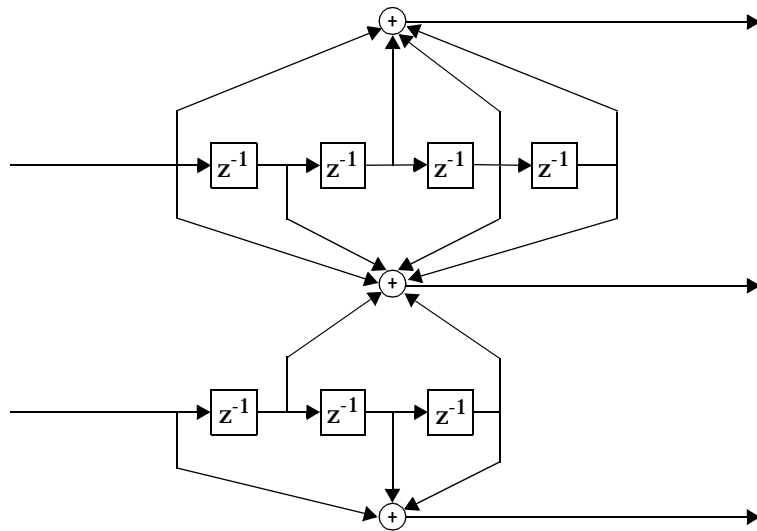
Examples of Convolutional Coding

The model below uses a rate 2/3 feedforward encoder, and the accompanying description explains how to determine the coding blocks' parameters from a schematic of the encoder. It also illustrates the use of the Error Rate Calculation block with a receive delay.

Another example using blocks from the Convolutional Coding library is the model in "Getting Started with the Communications Blockset" on page 1-1, which uses a quantizer and the Viterbi Decoder block to implement soft-decision Viterbi decoding. This model also illustrates the use of the Error Rate Calculation block with a receive delay.

Example: A Rate 2/3 Feedforward Convolutional Encoder

This example uses the rate 2/3 feedforward convolutional encoder depicted in the figure below.



How to Determine Coding Parameters. The Convolutional Encoder and Viterbi Decoder blocks can implement this code if their parameters have the appropriate values.

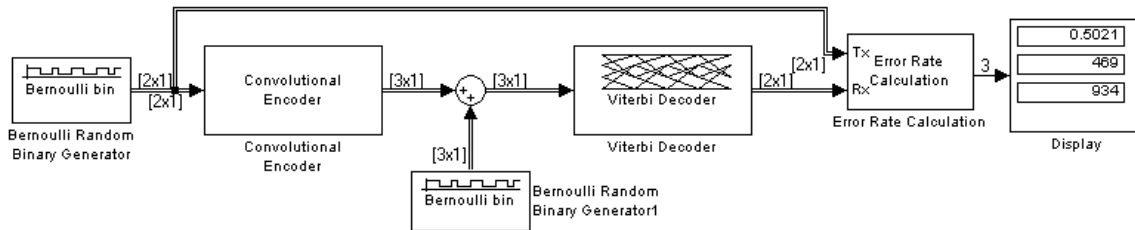
The encoder's constraint length is a vector of length 2 since the encoder has two inputs. The elements of this vector indicate the number of bits stored in each

shift register, including the current input bits. Counting memory spaces in each shift register in the diagram and adding one for the current inputs leads to a constraint length of [5 4].

To determine the code generator parameter as a 2-by-3 matrix of octal numbers, use the element in the i th row and j th column to indicate how the i th input contributes to the j th output. For example, to compute the element in the second row and third column, notice that the leftmost and two rightmost elements in the second shift register of the diagram feed into the sum that forms the third output. Capture this information as the binary number 1011, which is equivalent to the octal number 13. The full value of the code generator matrix is [27 33 0; 0 5 13].

To use the constraint length and code generator parameters in the Convolutional Encoder and Viterbi Decoder blocks, use the `poly2trellis` function to convert those parameters into a trellis structure.

How to Simulate the Encoder. Below is a model that simulates this encoder.



To open the completed model, click [here](#) in the MATLAB Help browser. To build the model, gather and configure these blocks:

- Bernoulli Random Binary Generator, in the Comm Sources library
 - Set **Probability of a zero** to . 5
 - Set **Initial seed** to any positive integer scalar
 - Set **Sample time** to . 5
 - Check the **Frame-based outputs** check box
 - Set **Samples per frame** to 2
- Convolutional Encoder
 - Set **Trellis structure** to `poly2trellis([5 4], [27 33 0; 0 5 13])`

- An additional copy of the Bernoulli Random Binary Generator, in the Comm Sources library
 - Set **Probability of a zero** to .05
 - Set **Initial seed** to any positive integer scalar
 - Set **Sample time** to 1/3
 - Check the **Frame-based outputs** check box
 - Set **Samples per frame** to 3
- Sum, in the Simulink Math library, with default parameter values
- Viterbi Decoder
 - Set **Trellis structure** to `poly2trellis([5 4], [27 33 0; 0 5 13])`
 - Set **Decision type** to **Hard Decision**
- Error Rate Calculation, in the Comm Sinks library
 - Set **Receive delay** to 68
 - Set **Output data to Port**
- Display, in the Simulink Sinks library
 - Drag the bottom edge of the icon to make the display big enough for three entries

Connect the blocks as in the figure, using the first Bernoulli Random Binary Generator block at the far left. Also, from the model window's **Simulation menu**, choose **Simulation parameters**; then in the **Simulation Parameters** dialog box, set **Stop time** to 500.

Notes on the Model. The matrix size annotations appear on the connecting lines only if you select **Signal Dimensions** from the model's **Format** menu. Notice that the encoder accepts a 2-by-1 frame-based vector and produces a 3-by-1 frame-based vector, while the decoder does the opposite. The **Samples per frame** parameter in the first Bernoulli Random Binary Generator block is 2 because the block must generate a message word of length 2, while that parameter in the second Bernoulli Random Binary Generator block is 3 because the block must perturb a codeword of length 3.

Also notice that the **Receive delay** parameter in the Error Rate Calculation block is 68, which is the vector length (2) of the recovered message times the **Traceback depth** value (34) in the Viterbi Decoder block. If you examined the transmitted and received signals as matrices in the MATLAB workspace, then you would see that the first 34 rows of the recovered message consists of zeros,

while subsequent rows are the decoded messages. Thus the delay in the received signal is 34 vectors of length 2, or 68 samples.

Running the model produces display output similar to that shown in the figure. The three numbers in the Display block indicate the error rate, the total number of errors, and the total number of comparisons that the Error Rate Calculation block made during the simulation. (The first two numbers vary depending on your **Initial seed** value in the Bernoulli Random Binary Generator blocks.) Because the simulation runs from time 0 to time 500, the block compares 501 pairs of vectors of length 2, hence 1002 pairs of samples. However, 68 of the received samples constitute the initial delay, so the total number of comparisons by the end of the simulation is 934.

Selected Bibliography for Convolutional Coding

- [1] Benedetto, Sergio and Guido Montorsi. "Performance of Continuous and Blockwise Decoded Turbo Codes." *IEEE Communications Letters*, vol. 1, May 1997. 77-79.
- [2] Benedetto, S., G. Montorsi, D. Divsalar, and F. Pollara. "A Soft-Input Soft-Output Maximum A Posterior (MAP) Module to Decode Parallel and Serial Concatenated Codes." *JPL TMO Progress Report*, vol. 42-127, November 1996. [This electronic journal is available at http://tmo.jpl.nasa.gov/tmo/progress_report/index.html.]
- [3] Clark, George C. Jr. and J. Bibb Cain. *Error-Correction Coding for Digital Communications*. New York: Plenum Press, 1981.
- [4] Gitlin, Richard D., Jeremiah F. Hayes, and Stephen B. Weinstein. *Data Communications Principles*. New York: Plenum, 1992.
- [5] Viterbi, Andrew J. "An Intuitive Justification and a Simplified Implementation of the MAP Decoder for Convolutional Codes." *IEEE Journal on Selected Areas in Communications*, vol. 16, February 1998. 260-264.

Interleaving

An interleaver permutes symbols according to a mapping. A corresponding deinterleaver uses the inverse mapping to restore the original sequence of symbols. Interleaving and deinterleaving can be useful for reducing errors caused by burst errors in a communication system.

You can open the Interleaving library by double-clicking on its icon in the main Communications Blockset library (`commlib`), or by typing

```
comminterleave2
```

at the MATLAB prompt.

Then you can open the interleaving sublibraries by double-clicking on their icons in the Interleaving library, or by typing these commands at the MATLAB prompt.

```
commblkintlv2  
commcnvintlv2
```

Interleaving Features of the Blockset

This blockset provides interleavers in two broad categories:

- Block interleavers. This category includes matrix, random, algebraic, and helical scan interleavers as special cases.
- Convolutional interleavers. This category includes a helical interleaver as a special case, as well as a general multiplexed interleaver.

In typical usage of all interleaver/deinterleaver pairs in this blockset, the parameters of the deinterleaver match those of the interleaver.

For background information about interleavers, see the works listed in “Selected Bibliography for Interleaving” on page 2-51.

Block Interleavers

A block interleaver accepts a set of symbols and rearranges them, without repeating or omitting any of the symbols in the set. The number of symbols in each set is fixed for a given interleaver. The interleaver’s operation on a set of symbols is independent of its operation on all other sets of symbols.

Types of Block Interleavers

The set of block interleavers in this library includes a general interleaver/deinterleaver pair as well as several special cases. Each special-case block uses the same computational code that its more general counterpart uses, but provides an interface that is more suitable for the special case.

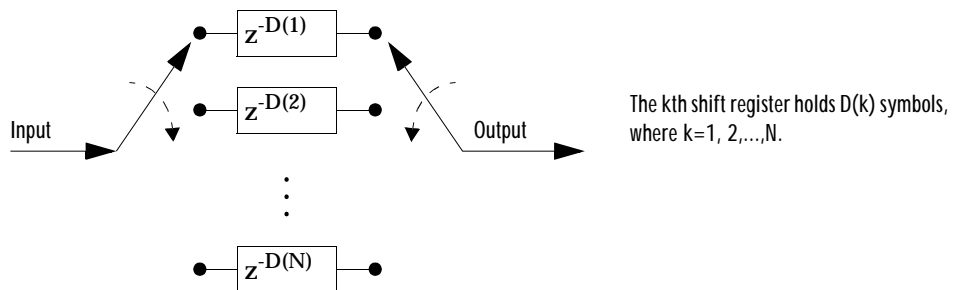
The Matrix Interleaver block accomplishes block interleaving by filling a matrix with the input symbols row by row and then sending the matrix contents to the output port column by column. For example, if the interleaver uses a 2-by-3 matrix to do its internal computations, then for an input of [1 2 3 4 5 6] the block produces an output of [1 4 2 5 3 6].

The Random Interleaver block chooses a permutation table randomly using the **Initial seed** parameter that you provide in the block mask. By using the same **Initial seed** value in the corresponding Random Deinterleaver block, you can restore the permuted symbols to their original ordering.

The Algebraic Interleaver block uses a permutation table that is algebraically derived. It supports Takeshita-Costello interleavers and Welch-Costas interleavers. These interleavers are described in [4].

Convolutional Interleavers

A convolutional interleaver consists of a set of shift registers, each with a fixed delay. In a typical convolutional interleaver, the delays are nonnegative integer multiples of a fixed integer (although a general multiplexed interleaver allows arbitrary delay values). Each new symbol from the input signal feeds into the next shift register and the oldest symbol in that register becomes part of the output signal. The schematic below depicts the structure of a convolutional interleaver by showing the set of shift registers and their delay values $D(1), D(2), \dots, D(N)$. The blocks in this library have mask parameters that indicate the delay for each shift register. The delay is measured in samples.



This section discusses:

- The types of convolutional interleavers included in the library
- The delay between the original sequence and the restored sequence
- An example that uses a convolutional interleaver

Types of Convolutional Interleavers

The set of convolutional interleavers in this library includes a general interleaver/deinterleaver pair as well as several special cases. Each special-case block uses the same computational code that its more general counterpart uses, but provides an interface that is more suitable for the special case.

The most general block in this library is the General Multiplexed Interleaver block, which allows arbitrary delay values for the set of shift registers. To implement the schematic above using this block, you would use an **Interleaver delay** parameter of $[D(1); D(2); \dots; D(N)]$.

More specific is the Convolutional Interleaver block, in which the delay value for the k th shift register is $(k-1)$ times the block's **Register length step** parameter. The number of shift registers in this block is the value of the **Rows of shift registers** parameter.

Finally, the Helical Interleaver block supports a special case of convolutional interleaving that fills an array with symbols in a helical fashion and empties the array row by row. To configure this interleaver, use the **Number of columns of helical array** parameter to set the width of the array, and use the **Group size** and **Helical array step size** parameters to determine how symbols

are placed in the array. See the reference page for the Helical Interleaver block for more details and an example.

Delay Between Original and Restored Sequences

After a sequence of symbols passes through a convolutional interleaver and a corresponding convolutional deinterleaver, the restored sequence lags behind the original sequence. The delay, measured in symbols, between the original and restored sequences is

$$(\text{Number of shift registers}) * (\text{Maximum delay among all shift registers})$$

for the most general multiplexed interleaver. If your model incurs an additional delay between the interleaver output and the deinterleaver input, then the restored sequence lags behind the original sequence by the sum of the additional delay and the amount in the formula above.

Note For proper synchronization, the delay in your model between the interleaver output and the deinterleaver input must be an integer multiple of the number of shift registers. You can use the Integer Delay block in the DSP Blockset to adjust delays manually, if necessary.

Convolutional Interleaver block. In the special case implemented by the Convolutional Interleaver/Convolutional Deinterleaver pair, note that the number of shift registers is the **Rows of shift registers** parameter, while the maximum delay among all shift registers is

$$(\text{Register length step}) * (\text{Rows of shift registers} - 1)$$

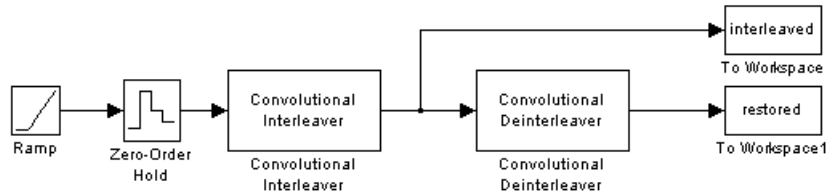
Helical Interleaver block. In the special case implemented by the Helical Interleaver/Helical Deinterleaver pair, the delay between the restored sequence and the original sequence is

$$CN \left\lceil \frac{s(C-1)}{N} \right\rceil$$

where C is the **Number of columns in helical array** parameter, N is the **Group size** parameter, and s is the **Helical array step size** parameter.

Example: Convolutional Interleavers

The example below illustrates convolutional interleaving and deinterleaving using a sequence of consecutive integers. It also illustrates the inherent delay and the effect of the interleaving blocks' initial conditions.



To open the completed model, click [here](#) in the MATLAB Help browser. To build the model, gather and configure these blocks:

- Ramp, in the Simulink Sources library. Use default parameters.
- Zero-Order Hold, in the Simulink Discrete library. Use default parameters.
- Convolutional Interleaver
 - Set **Rows of shift registers** to 3
 - Set **Initial conditions** to `[-1 -2 -3]'`
- Convolutional Deinterleaver
 - Set **Rows of shift registers** to 3
 - Set **Initial conditions** to `[-1 -2 -3]'`
- Two copies of To Workspace, in the Simulink Sinks library
 - Set **Variable name** to `interleaved` and `restored`, respectively, in the two copies of this block
 - Set **Save format** to **Array** in each of the two copies of this block

Connect the blocks as shown above. Also, from the model window's **Simulation** menu, choose **Simulation parameters**; then in the **Simulation Parameters** dialog box, set **Stop time** to 20. Run the simulation, then execute this command.

```
comparison = [[0:20]', interleaved, restored]
```

compari son =

0	0	- 1
1	- 2	- 2
2	- 3	- 3
3	3	- 1
4	- 2	- 2
5	- 3	- 3
6	6	- 1
7	1	- 2
8	- 3	- 3
9	9	- 1
10	4	- 2
11	- 3	- 3
12	12	0
13	7	1
14	2	2
15	15	3
16	10	4
17	5	5
18	18	6
19	13	7
20	8	8

In this output, the first column contains the original symbol sequence. The second column contains the interleaved sequence, while the third column contains the restored sequence.

The negative numbers in the interleaved and restored sequences come from the interleaving blocks' initial conditions, not from the original data. The first of the original symbols appears in the restored sequence only after a delay of 12 symbols. The delay of the interleaver-deinterleaver combination is the product of the number of shift registers (3) and the maximum delay among all shift registers (4).

Selected Bibliography for Interleaving

[1] Berlekamp, E. R. and P. Tong. "Improved Interleavers for Algebraic Block Codes." U. S. Patent 4559625, Dec. 17, 1985.

- [2] Clark, George C. Jr. and J. Bibb Cain. *Error-Correction Coding for Digital Communications*. New York: Plenum Press, 1981.
- [3] Forney, G., D., Jr. "Burst-Correcting Codes for the Classic Bursty Channel." *IEEE Transactions on Communications*, vol. COM-19, October 1971. 772-781.
- [4] Heegard, Chris and Stephen B. Wicker. *Turbo Coding*. Boston: Kluwer Academic Publishers, 1999.
- [5] Ramsey, J. L. "Realization of Optimum Interleavers." *IEEE Transactions on Information Theory*, IT-16 (3), May 1970. 338-345.
- [6] Takeshita, O. Y. and D. J. Costello, Jr. "New Classes Of Algebraic Interleavers for Turbo-Codes." *Proc. 1998 IEEE International Symposium on Information Theory*, Boston, Aug. 16-21, 1998. 419.

Analog Modulation

In most media for communication, only a fixed range of frequencies is available for transmission. One way to communicate a message signal whose frequency spectrum does not fall within that fixed frequency range, or one that is otherwise unsuitable for the channel, is to alter a transmittable signal according to the information in your message signal. This alteration is called *modulation*, and it is the modulated signal that you transmit. The receiver then recovers the original signal through a process called *demodulation*. This section describes how to modulate and demodulate analog signals with the Communications Blockset. After giving instructions for accessing the analog modulation blocks, it goes on to discuss these topics:

- “Analog Modulation Features of the Blockset” on page 2-53
- “Baseband Modulated Signals Defined” on page 2-54
- “Representing Signals for Analog Modulation” on page 2-55
- “Timing Issues in Analog Modulation” on page 2-55
- “Filter Design Issues” on page 2-59

Accessing Analog Modulation Blocks

You can open the Modulation library by double-clicking on its icon in the main Communications Blockset library (comm_l b), or by typing

```
commmod2
```

at the MATLAB prompt.

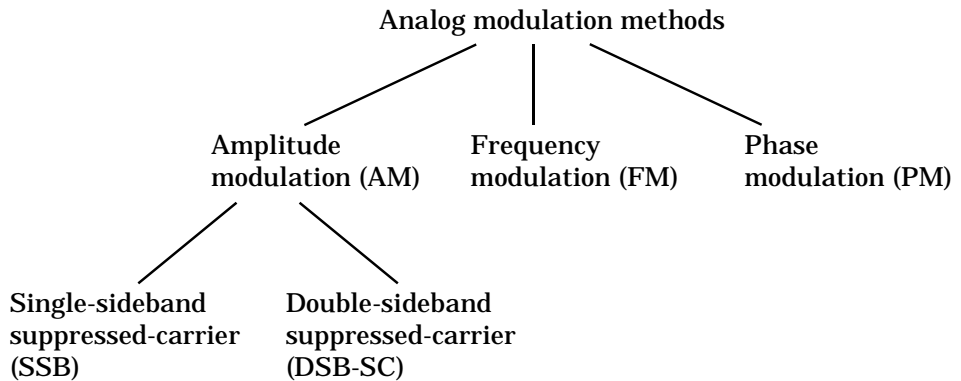
Then you can open the Analog Baseband and Analog Passband sublibraries by double-clicking on their icons in the Modulation library, or by typing these commands at the MATLAB prompt.

```
commanabbnd2
```

```
commanapbnd2
```

Analog Modulation Features of the Blockset

The figure below shows the modulation techniques that the Communications Blockset supports for analog signals. As the figure suggests, some categories of techniques include named special cases.



For a given modulation technique, two ways to simulate modulation techniques are called *baseband* and *passband*. Baseband simulation, also known as the *lowpass equivalent method*, requires less computation. This blockset supports both baseband and passband simulation. Since baseband simulation is more prevalent, this guide focuses more on baseband simulation.

The modulation and demodulation blocks also let you control such features as the initial phase of the modulated signal and post-demodulation filtering.

Baseband Modulated Signals Defined

A baseband representation of a modulated signal is often more convenient for simulation than the passband representation is. Suppose the modulated signal has the waveform

$$Y_1(t) \cos(2\pi f_c t + \theta) - Y_2(t) \sin(2\pi f_c t + \theta)$$

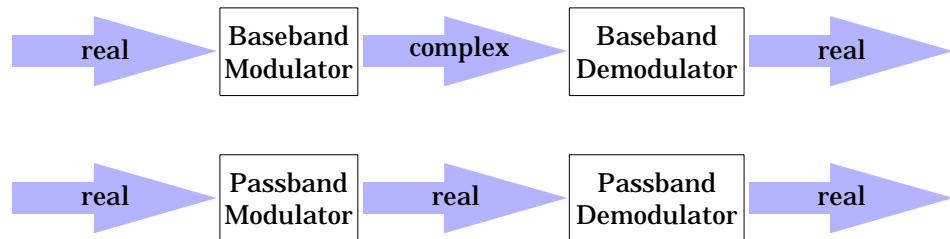
where Y_1 and Y_2 are amplitude terms, f_c is the carrier frequency and θ is the carrier signal's initial phase. A baseband simulation recognizes that this equals

$$\operatorname{Re} \left[(Y_1(t) + jY_2(t)) e^{j\theta} e^{j2\pi f_c t} \right]$$

and models only the part inside the curly brackets. Here j is the square root of -1. The complex vector y is a sampling of the complex signal $(Y_1(t) + j Y_2(t)) e^{j\theta}$.

Representing Signals for Analog Modulation

Analog modulation blocks in this blockset process only sample-based scalar signals. The data types of inputs and outputs are depicted in the figure below.



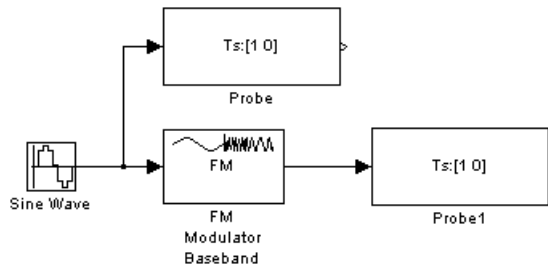
Timing Issues in Analog Modulation

A few timing issues are important for simulating analog modulation with this blockset. The sections below illustrate choices of signal sample times and simulation step sizes.

Signal Sample Times

All analog demodulators in this blockset produce discrete-time, not continuous-time, output. These blocks require you to specify the output sample time as a mask parameter. In addition, some analog modulators require you to specify the sample time as a mask parameter. Modulators in this category are FM Modulator Baseband, FM Modulator Passband, SSB AM Modulator Baseband, and SSB AM Modulator Passband.

Example Using a Modulator. In the figure below, both Probe blocks show a sample time of 1 second in their icons. (The display Ts: [1 0] indicates a sample time of 1 second and a sample time offset of 0.) Setting the **Sample time** parameter in the FM Modulator Baseband block to 1 is appropriate because the input to this block also has a sample time of 1 second.



To open the completed model, click [here](#) in the MATLAB Help browser. To build the model, gather and configure these blocks:

- Sine Wave, in the Simulink Sources library (*not* the Sine Wave block in the DSP Blockset DSP Sources library)
 - Set **Sample time** to 1
- FM Modulator Baseband, in the Analog Baseband sublibrary of the Modulation library
 - Set **Sample time** to 1
- Two copies of Probe, in the Simulink Signals & Systems library
 - Uncheck all boxes except **Probe sample time**

Connect the blocks as in the figure. Then select **Update diagram** from the model window's **Edit** menu, which updates the display on each Probe block's icon. (Running the model is not instructive because it does not represent a complete system.)

Simulation Step Sizes

If you use passband modulation with continuous-time signals, then you need to set the simulation step size, based on the carrier frequency. By the Nyquist theorem, the simulation sampling rate must be at least twice as large as the modulation carrier frequency. Equivalently, the simulation step size must be no larger than half the modulation carrier period.

When you begin a new model, Simulink automatically determines the default step size. To change the step size from the default to a different value, use this procedure:

- 1 Select **Simulation parameters** from the model window's **Simulation** menu

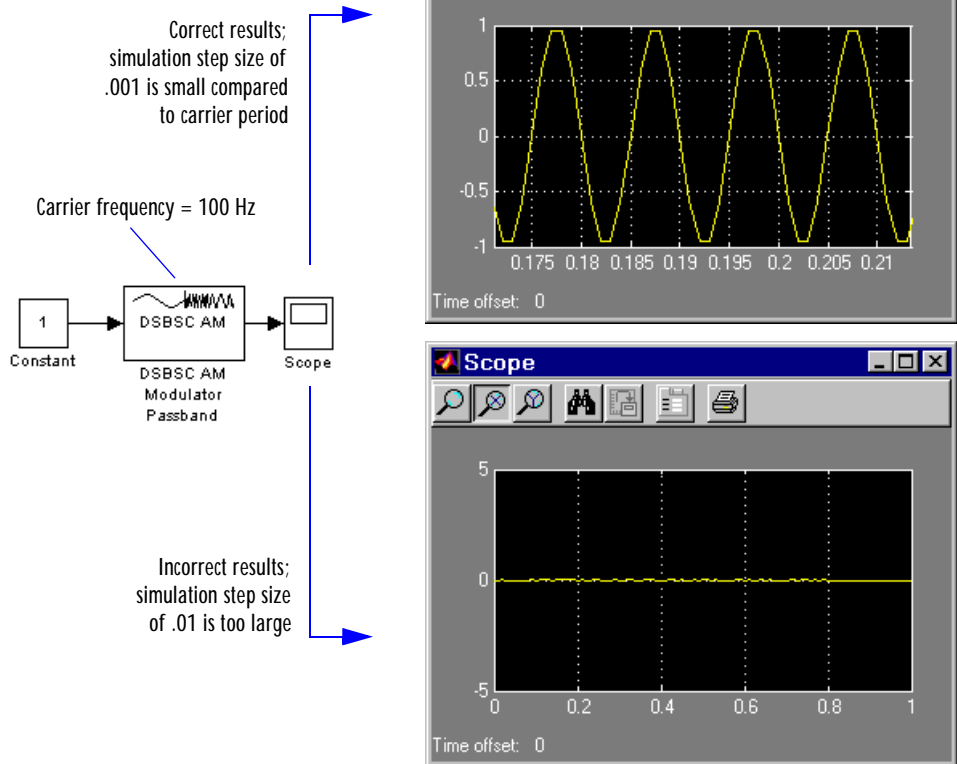
- 2 Select the **Solver** panel
- 3 Set the **Max step size** and **Initial step size** parameters to numerical values (that is, not auto) that are appropriate for your model.

In some situations, Simulink automatically corrects a faulty simulation step size. For example, if a signal in your model has a sample time of .1 second and you set the model's **Max step size** parameter to 1, then running the model produces this response in the command window.

Warning: Maximum step size (1) is larger than the fastest discrete sampling period (0.1) time. Setting maximum step size to 0.1.

Example Using Step Size Relative to Carrier Period. The model below illustrates why the simulation step size in a passband simulation must be appropriate for a given carrier frequency. The first Scope image shows the correct result of modulating a constant signal using double-sideband suppressed-carrier amplitude modulation, while the second Scope image shows incorrect results. The incorrect results occur because the simulation step size is too large relative to the modulation carrier frequency.

In this case, the DSBSC AM Modulator Passband block uses a **Carrier frequency** parameter of 100. That is, the carrier's period is .01 second and an appropriate simulation step size is no larger than .005. Therefore, a simulation step size of .01 second is too large to satisfy the Nyquist criterion. However, a simulation step size of .001 second is sufficiently small.



To open the completed model, click here in the MATLAB Help browser. To build the model, gather these blocks with their default parameters:

- Constant, in the Simulink Sources library
- DSBSC AM Modulator Passband, in the Analog Passband sublibrary of the Modulation library
- Scope, in the Simulink Sinks library

Connect the blocks as in the figure. Also, from the model window's **Simulation** menu, choose **Simulation parameters**; then in the **Simulation Parameters** dialog box, set **Stop time** to 1.

To generate the correct results as in the first Scope image in the figure, return to the **Simulation Parameters** dialog box and set both the **Max step size** and **Initial step size** parameters to .001. Then run the model and use the Scope window's zooming tools to study the sinusoidal output curve. You can also generate incorrect results as in the second Scope image in the figure, by changing the **Max step size** and **Initial step size** parameters to .01 and running the model again.

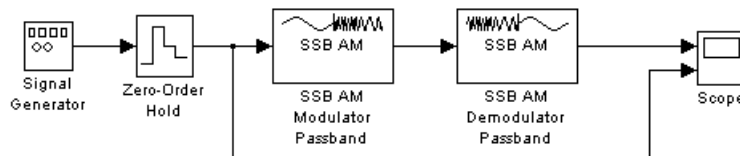
Filter Design Issues

After demodulating, you might want to filter out the carrier signal, especially if you are using passband simulation. The Signal Processing Toolbox provides functions that can help you design your filter, such as `butter`, `cheby1`, `cheby2`, and `ellip`. Different demodulation methods have different properties, and you might need to test your application with several filters before deciding which is most suitable. This section mentions two issues that relate to the use of filters: cutoff frequency and time lag.

Example: Varying the Filter's Cutoff Frequency

In many situations, a suitable cutoff frequency is half the carrier frequency. Since the carrier frequency must be higher than the bandwidth of the message signal, a cutoff frequency chosen in this way limits the bandwidth of the message signal. If the cutoff frequency is too high, then the carrier frequency may not be filtered out. If the cutoff frequency is too low, then it might narrow the bandwidth of the message signal.

The example below modulates a sawtooth message signal, demodulates the resulting signal using a Butterworth filter, and plots the original and recovered signals. The Butterworth filter is implemented within the SSB AM Demodulator Passband block.



Before building the model, first execute this command at the MATLAB prompt.

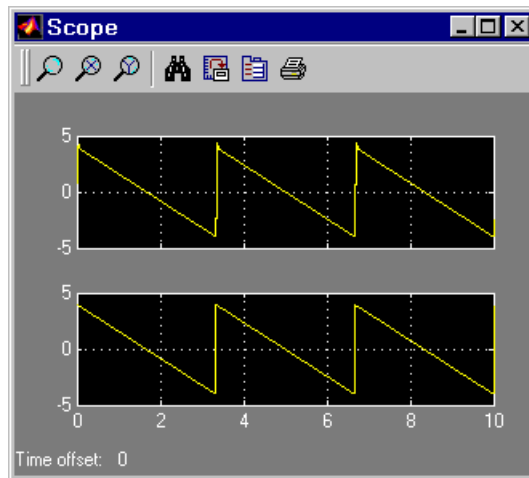
```
[num, den] = butter(2, 25*.01); % Design Butterworth filter.
```

Here, 2 is the order of the Butterworth filter, 25 is the carrier signal frequency and .01 is the sample time of the signal in Simulink. The variables num and den represent the numerator and denominator, respectively, of the filter's transfer function. These variables reside in the MATLAB workspace, where Simulink can access them during the simulation. The `butter` function is in the Signal Processing Toolbox.

Now to open the completed model, click [here](#) in the MATLAB Help browser. To build the model, gather and configure these blocks:

- Signal Generator, in the Simulink Sources library
 - Set **Wave form** to **Sawtooth**
 - Set **Amplitude** to 4
 - Set **Frequency** to . 3
- Zero-Order Hold, in the Simulink Discrete library
 - Set **Sample time** to . 01
- SSB AM Modulator Passband, in the Analog Passband sublibrary of the Modulation library
 - Set **Carrier frequency** to 25
 - Set **Time delay for Hilbert transform filter** to . 1
 - Set **Sample time** to . 01
- SSB AM Demodulator Passband, in the Analog Passband sublibrary of the Modulation library
 - Set **Carrier frequency** to 25
 - Set **Lowpass filter numerator** to num
 - Set **Lowpass filter denominator** to den
 - Set **Sample time** to . 01
- Scope, in the Simulink Sinks library
 - After double-clicking on the block to open it, click on the **Properties** icon and set **Number of axes** to 2

Connect the blocks as in the figure. Running the model produces the scope image below. The image reflects the original and recovered signals, with a moderate filter cutoff.



Other Filter Cutoffs. To see the effect of a lowpass filter with a *higher* cutoff frequency, type

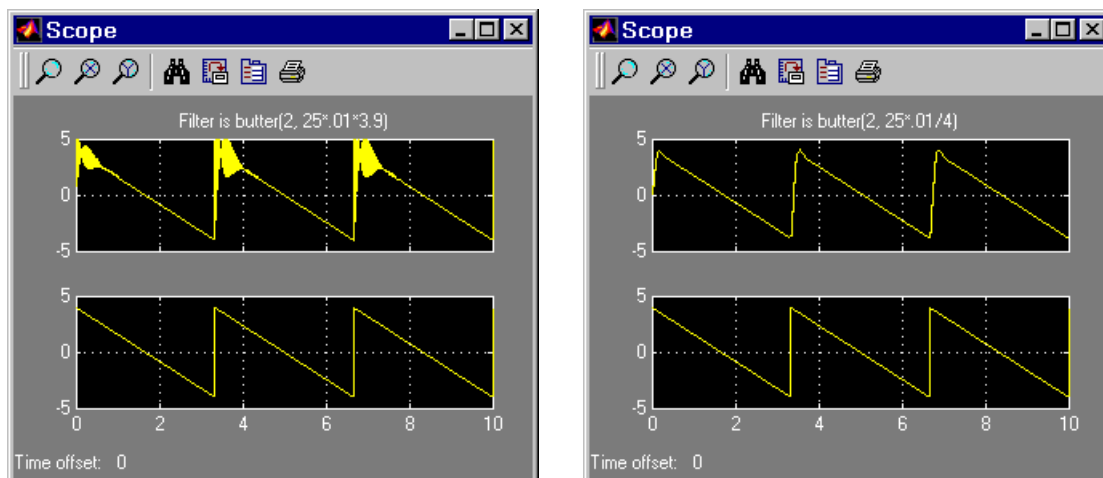
```
[num, den] = butter(2, 25*.01*3.9); % Design Butterworth filter.
```

at the MATLAB prompt and then run the simulation again. The new result is the left image in the figure below. The higher cutoff frequency allows the carrier signal to interfere with the demodulated signal.

To see the effect of a lowpass filter with a *lower* cutoff frequency, type

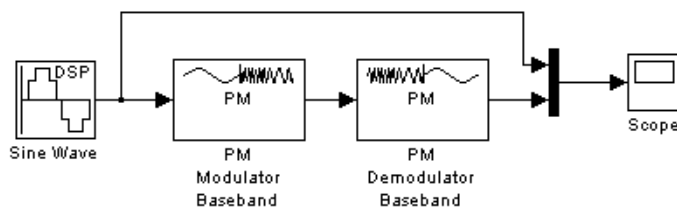
```
[num, den] = butter(2, 25*.01/4); % Design Butterworth filter.
```

at the MATLAB prompt and then run the simulation again. The new result is the right image in the figure below. The lower cutoff frequency narrows the bandwidth of the demodulated signal.



Example: Time Lag from Filtering

There is invariably a delay between a demodulated signal and the original received signal. Both the filter order and the filter parameters directly affect the length of this delay. The example below illustrates the delay by plotting a signal before modulation and after demodulation. The curve with amplitude one is the original sine wave and the other curve is the recovered signal.

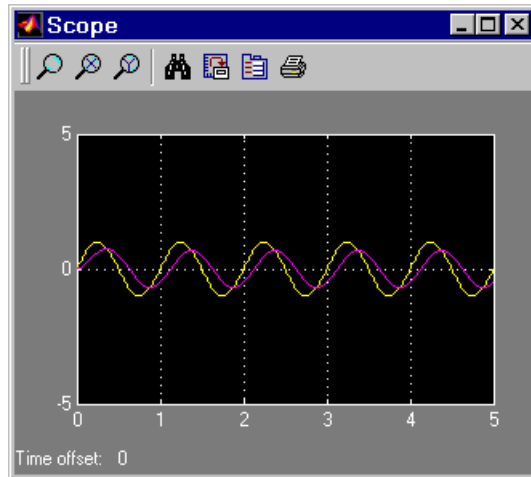


To open the completed model, click [here](#) in the MATLAB Help browser. To build the model, gather and configure these blocks:

- Sine Wave, in the DSP Blockset DSP Sources library (*not* the Sine Wave block in the Simulink Sources library)
 - Set **Frequency** to 1

- PM Modulator Baseband, in the Analog Baseband sublibrary of the Modulation library. Use default parameters.
- PM Demodulator Baseband, in the Analog Baseband sublibrary of the Modulation library. Use default parameters.
- Mux, in the Simulink Signals & Systems library
- Scope, in the Simulink Sinks library

Connect the blocks as in the figure. Also, from the model window's **Simulation** menu, choose **Simulation parameters**; then in the **Simulation Parameters** dialog box, set **Stop time** to 5. Running the model produces the scope image below.



Digital Modulation

Like analog modulation, digital modulation alters a transmittable signal according to the information in a message signal. However, in this case, the message signal is a discrete-time signal that can assume finitely many values. This section describes how to modulate and demodulate digital signals with the Communications Blockset. After giving instructions for accessing the digital modulation blocks, it goes on to discuss these topics:

- “Digital Modulation Features of the Blockset” on page 2-65
- “Representing Signals for Digital Modulation” on page 2-68
- “Delays in Digital Modulation” on page 2-69
- “Upsampled Signals and Rate Changes” on page 2-72
- “Examples of Digital Modulation” on page 2-75

For background material on the subject of digital modulation, see the works listed in “Selected Bibliography for Digital Modulation” on page 2-83.

Accessing Digital Modulation Blocks

You can open the Modulation library by double-clicking on its icon in the main Communications Blockset library (`comm1` b), or by typing

```
commmod2
```

at the MATLAB prompt.

Then you can open the Digital Baseband and Digital Passband sublibraries by double-clicking on their icons in the Modulation library, or by typing these commands at the MATLAB prompt.

```
commdi gbbnd2
```

```
commdi gpbd2
```

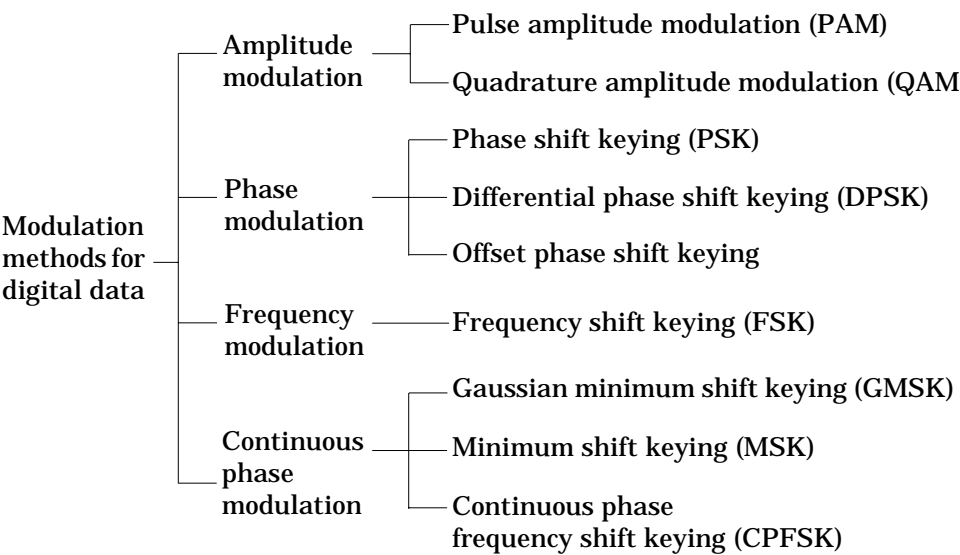
The Digital Baseband and Digital Passband libraries have sublibraries of their own. You can open each of these sublibraries by double-clicking on the icon listed in the table below, or by typing its name at the MATLAB prompt.

Table 2-1: Sublibraries of Digital Baseband and Digital Passband

Kind of Modulation	Icon in Digital Baseband or Digital Passband Library	Name of Sublibrary Model
Amplitude modulation	AM	commdi gbbndam2, commdi gpbndam2
Phase modulation	PM	commdi gbbndpm2, commdi gpbndpm2
Frequency modulation	FM	commdi gbbndfm2, commdi gpbndfm2
Continuous phase modulation	CPM	commdi gbbndcpm2, commdi gpbndcpm2

Digital Modulation Features of the Blockset

The figure below shows the modulation techniques that the Communications Blockset supports for digital data. All of the methods at the far right are implemented in both passband and baseband blocks.



General and Specific Modulation Methods

Some digital modulation sublibraries contain blocks that implement special cases of a more general technique and are, in fact, special cases of a more general block. These special-case blocks use the same computational code that their general counterparts use, but provide an interface that is either simpler or more suitable for the special case. The table below lists special-case modulators, their general counterparts, and the conditions under which the two are equivalent. The situation is analogous for demodulators.

Table 2-2: General and Specific Blocks

General Modulator	Specific Modulator	Specific Conditions
General QAM Modulator Baseband, General QAM Modulator Passband	Rectangular QAM Modulator Baseband, Rectangular QAM Modulator Passband	Predefined constellation containing 2^K points on a rectangular lattice

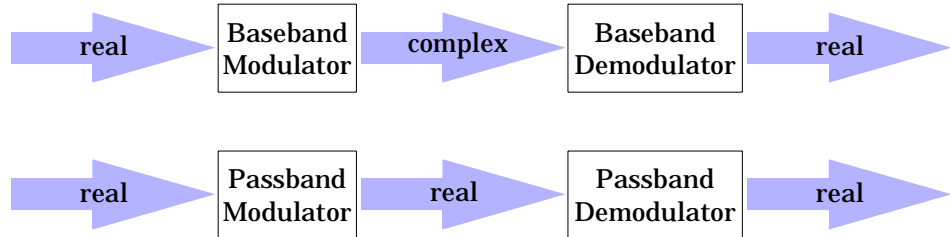
Table 2-2: General and Specific Blocks

General Modulator	Specific Modulator	Specific Conditions
M-PSK Modulator Baseband	BPSK Modulator Baseband	M-ary number parameter is 2.
	QPSK Modulator Baseband	M-ary number parameter is 4.
M-DPSK Modulator Baseband	DBPSK Modulator Baseband	M-ary number parameter is 2.
	DQPSK Modulator Baseband	M-ary number parameter is 4.
CPM Modulator Baseband, CPM Modulator Passband	GMSK Modulator Baseband, GMSK Modulator Passband	M-ary number parameter is 2, Frequency pulse shape parameter is Gaussian .
	MSK Modulator Baseband, MSK Modulator Passband	M-ary number parameter is 2, Frequency pulse shape parameter is Rectangular , Pulse length parameter is 1.
	CPFSK Modulator Baseband, CPFSK Modulator Passband	Frequency pulse shape parameter is Rectangular , Pulse length parameter is 1.

Furthermore, the CPFSK Modulator Baseband and CPFSK Modulator Passband blocks are similar to the M-FSK Modulator Baseband and M-FSK Modulator Passband blocks, respectively, when the M-FSK blocks use continuous phase transitions. However, the M-FSK features of this blockset differ from the CPFSK features in their mask interfaces and in the demodulator implementations.

Representing Signals for Digital Modulation

All digital modulation blocks process only discrete-time signals. The data types of inputs and outputs are depicted in the figure below.



Note If you are simulating baseband modulation using a method that produces real-valued data (such as pulse amplitude modulation) and if you want the modulated signal to be a real Simulink signal instead of a complex Simulink signal, then you can use the corresponding passband block with a **Carrier frequency** parameter of zero.

Binary-Valued and Integer-Valued Signals

Some digital modulation blocks can accept either integers or binary representations of such integers. The corresponding demodulation blocks can output either integers or their binary representations. This section describes how modulation blocks process binary inputs; the case for demodulation blocks is the reverse.

If a modulator block's **Input type** parameter is set to **Bit**, then the block accepts binary representations of integers between 0 and M-1. It modulates each group of K bits, called a binary *word*. Also, these rules apply to the binary input mode:

- For baseband modulation, the input vector length must be an integer multiple of K. If the input is frame-based, then it must be a column vector.
- For passband modulation, the input must have length K and must be sample-based.

In binary input mode, the **Constellation ordering** (or **Symbol set ordering**, depending on the type of modulation) parameter indicates how the block maps a group of K input bits to a corresponding integer. If this parameter is set to **Binary**, then the block maps $[u(1) \ u(2) \ \dots \ u(K)]$ to the integer

$$\sum_{i=1}^K u(i)2^{K-i}$$

and subsequently behaves as if this integer were the input value. Notice that $u(1)$ is the most significant bit.

For example, if $M = 8$, **Constellation ordering** (or **Symbol set ordering**) is set to **Binary**, and the binary input word is $[1 \ 1 \ 0]$, then the block internally converts $[1 \ 1 \ 0]$ to the integer 6. The block produces the same output as in the case when the input is 6 and the **Input type** parameter is **Integer**.

If **Constellation ordering** (or **Symbol set ordering**) is set to **Gray**, then the block uses a Gray-coded arrangement. The explicit mapping is described in “Algorithm” on the reference page for the M-PSK Modulator Baseband block.

Delays in Digital Modulation

Digital modulation and demodulation blocks sometimes incur delays between their inputs and outputs, depending on their configuration and on properties of their signals. The table below lists sources of delay and the situations in which

they occur. If more than one situation applies to a given model, then the separate delays are additive.

Table 2-3: Delays Resulting from Digital Modulation or Demodulation

Modulation or Demodulation Type	Situation in Which Delay Occurs	Amount of Delay
All demodulators in AM, PM, and FM sublibraries, except OQPSK	Samples per symbol parameter is greater than 1, and the input is sample-based.	One output period
All demodulators in CPM sublibrary	Sample-based input, D = Traceback length parameter	D+1 output periods
	Frame-based input, D = Traceback length parameter	D output periods
All passband demodulators	Always	One output period
OQPSK Modulator Baseband, OQPSK Modulator Passband	Always	One output symbol, which is half the input period
OQPSK Demodulator Baseband, OQPSK Demodulator Passband	Always	Three input symbols, which is 3/2 times the output period

As a result of delays, data that enters a modulation or demodulation block at time T appears in the output at time T+delay. In particular, if your simulation computes error statistics or compares transmitted with received data, then it must take the delay into account when performing such computations or comparisons.

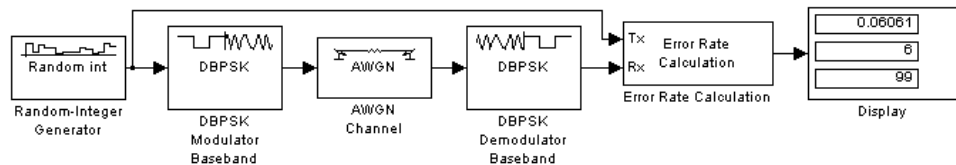
First Output Sample in DPSK Demodulation. In addition to the delays mentioned above, the DPSK, DQPSK, and DBPSK demodulators produce output whose

first sample is unrelated to the input. This is related to the differential modulation technique, not the particular implementation of it.

Example: Delays From Demodulation

Demodulation in the model below causes the demodulated signal to lag, compared to the unmodulated signal. This delay is typical for sample-based data that the modulator upsamples. When computing error statistics, the model accounts for the delay by setting the Error Rate Calculation block's **Receive delay** parameter to 1. If the **Receive delay** parameter had a different value, then the error rate showing at the top of the Display block would be close to 1/2.

Note If this model used the OQPSK method instead of DBPSK, then the proper **Receive delay** parameter would be 2 instead of 1.



To open the completed model, click [here](#) in the MATLAB Help browser. To build the model, gather and configure these blocks:

- Random-Integer Generator, in the Comm Sources library
 - Set **M-ary number** to 2
 - Set **Initial seed** to any positive integer scalar
- DBPSK Modulator Baseband, in the PM sublibrary of the Digital Baseband sublibrary of Modulation
 - Set **Samples per symbol** to 8
- AWGN Channel, in the Channels library
 - Set **Es/No** to 4
- DBPSK Demodulator Baseband, in the PM sublibrary of the Digital Baseband sublibrary of Modulation
 - Set **Samples per symbol** to 8

- Error Rate Calculation, in the Comm Sinks library
 - Set **Receive delay** to 1
 - Set **Computation delay** to 1
 - Set **Output data** to **Port**
- Display, in the Simulink Sinks library
 - Drag the bottom edge of the icon to make the display big enough for three entries

Connect the blocks as shown above. Also, from the model window’s **Simulation menu**, choose **Simulation parameters**; then in the **Simulation Parameters** dialog box, set **Stop time** to 100. Then run the model and observe the error rate at the top of the Display block’s icon. Your error rate will vary depending on your **Initial seed** value in the Random-Integer Generator block.

Upsampled Signals and Rate Changes

Digital baseband modulation blocks can output an upsampled version of the modulated signal, while digital baseband demodulation blocks can accept an upsampled version of the modulated signal as input. Each block’s **Samples per symbol** parameter, *S*, is the upsampling factor in both cases. It must be a positive integer. Depending on whether the signal is frame-based or sample-based, the block either changes the signal’s vector size or its sample time, as the table below indicates. Only the OQPSK blocks deviate from the information in the table, in that *S* is replaced by *2S* in the scaling factors.

Table 2-4: Processing of Upsampled Modulated Data (Except OQPSK Method)

Computation Type	Input Frame Status	Result
Modulation	Frame-based	Output vector length is <i>S</i> times the number of integers or binary words in the input vector. Output sample time equals the input sample time.
Modulation	Sample-based	Output vector is a scalar. Output sample time is 1/ <i>S</i> times the input sample time.

Table 2-4: Processing of Upsampled Modulated Data (Except OQPSK Method)

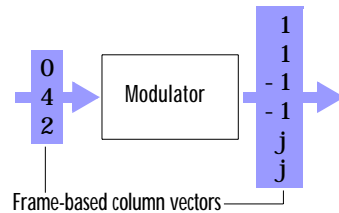
Computation Type	Input Frame Status	Result
Demodulation	Frame-based	Number of integers or binary words in the output vector is 1/S times the number of samples in the input vector. Output sample time equals the input sample time.
Demodulation	Sample-based	Output signal contains one integer or one binary word. Output sample time is S times the input sample time. Furthermore, if $S > 1$ and the demodulator is from the AM, PM, or FM sublibrary, then the demodulated signal is delayed by one output sample period. There is no delay if $S = 1$ or if the demodulator is from the CPM sublibrary.

Digital passband blocks can also process upsampled data, but only as an intermediate internal format. For more information about this, see the description of the **Baseband samples per symbol** parameter on the reference page for any digital passband modulation block. Also note that passband blocks process only sample-based data, not frame-based data.

Illustrations of Size or Rate Changes

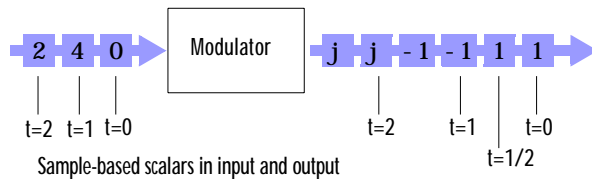
The schematics below illustrate how a baseband modulator (other than OQPSK) upsamples a triplet of frame-based and sample-based integers. In both cases, the **Samples per symbol** parameter is 2.

Frame-Based Upsampling



Output sample time = Input sample time
Output length = 2* (# of input integers)

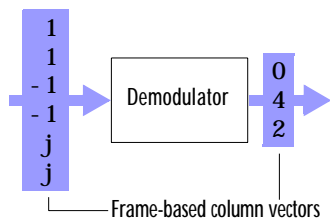
Sample-Based Upsampling



Output sample time = (Input sample time)/2
Output length = # of input integers

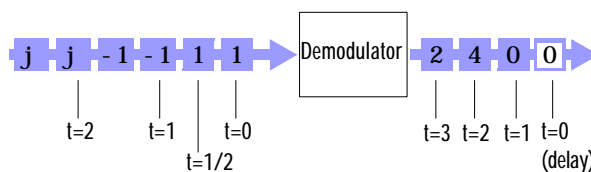
The schematics below illustrate how a demodulator (other than OQPSK or one from the CPM sublibrary) processes three doubly-sampled symbols using both frame-based and sample-based inputs. In both cases, the **Samples per symbol** parameter is 2. Notice that the sample-based schematic includes an output delay of one sample period.

Frame-Based Upsampled Input



Output sample time = Input sample time
Output length = (# of input samples)/2

Sample-Based Upsampled Input



Output sample time = 2*(Input sample time)
Scalar input and output
First output element represents delay

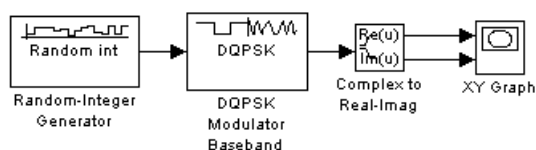
Examples of Digital Modulation

This section builds a few simple example models to illustrate the modulation methods and how the Communications Blockset allows you to implement them. The examples are:

- “DQPSK Signal Constellation Points and Transitions” on page 2-75
- “Rectangular QAM Modulation and Scatter Diagram” on page 2-76
- “Phase Tree for Continuous Phase Modulation” on page 2-78
- “Passband Digital Modulation” on page 2-81

DQPSK Signal Constellation Points and Transitions

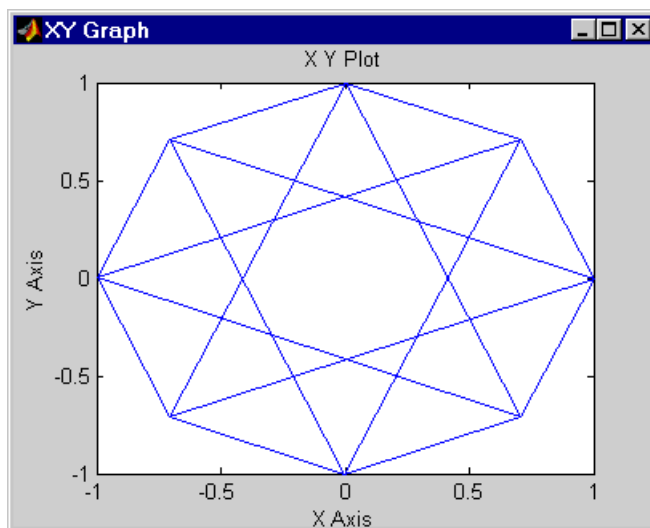
The model below plots the output of the DQPSK Modulator Baseband block. The image shows the possible transitions from each symbol in the DQPSK signal constellation to the next symbol.



To open the completed model, click [here](#) in the MATLAB Help browser. To build the model, gather and configure these blocks:

- Random-Integer Generator, in the Comm Sources library
 - Set **M-ary number** to 4
 - Set **Initial seed** to any positive integer scalar
 - Set **Sample time** to . 01
- DQPSK Modulator Baseband, in the PM sublibrary of the Digital Baseband sublibrary of Modulation
- Complex to Real-Imag, in the Simulink Math library
- XY Graph, in the Simulink Sinks library

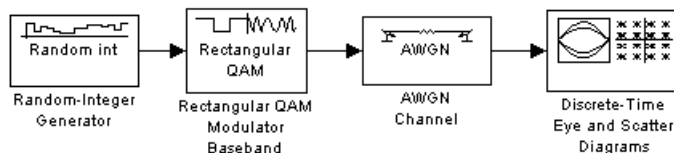
Use the blocks' default parameters unless otherwise instructed. Connect the blocks as in the figure. Running the model produces the plot below. The plot reflects the transitions among the eight DQPSK constellation points.



This plot illustrates $\pi/4$ -DQPSK modulation, because the default **Phase offset** parameter in the DQPSK Modulator Baseband block is $\pi/4$. To see how the phase offset influences the signal constellation, change the **Phase offset** parameter in the DQPSK Modulator Baseband block to $\pi/8$ or another value. Run the model again and observe how the plot changes.

Rectangular QAM Modulation and Scatter Diagram

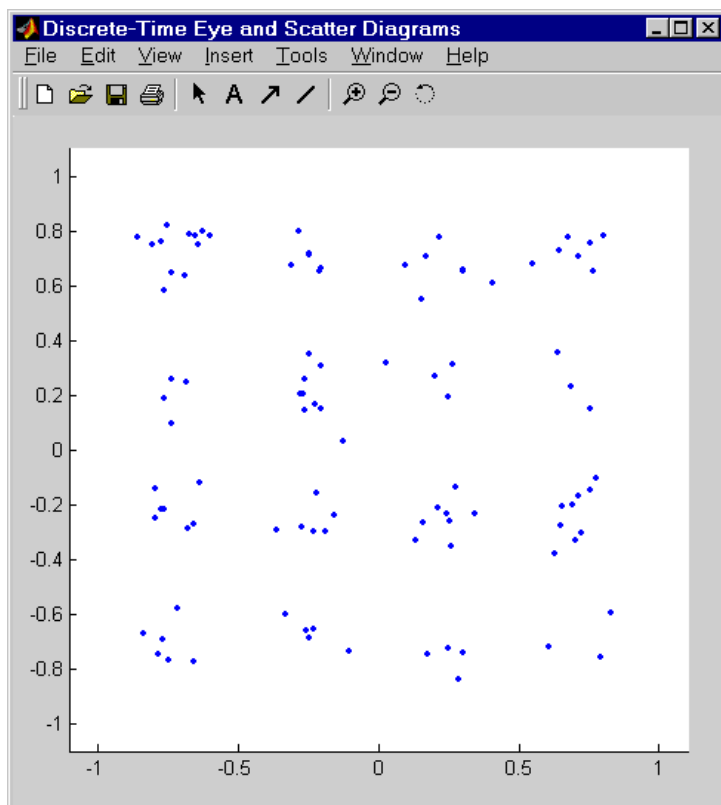
The model below uses the M-QAM Modulator Baseband block to modulate random data. After passing the symbols through a noisy channel, the model produces a scatter diagram of the noisy data. The diagram suggests what the underlying signal constellation looks like and shows that the noise distorts the modulated signal from the constellation.



To open the completed model, click [here](#) in the MATLAB Help browser. To build the model, gather and configure these blocks:

- Random-Integer Generator, in the Comm Sources library
 - Set **M-ary number** to 16
 - Set **Initial seed** to any positive integer scalar
 - Set **Sample time** to . 1
- Rectangular QAM Modulator Baseband, in the AM sublibrary of the Digital Baseband sublibrary of Modulation
 - Set **Normalization method** to **Peak Power**
- AWGN Channel, in the Channels library
 - Set **Es/No** to 20
 - Set **Symbol period** to . 1
- Discrete-Time Eye and Scatter Diagrams, in the Comm Sinks library
 - Set **Trace period** to . 1
 - Set **Decision point** to . 07
 - Set **Lower and upper bounds of diagram** to [- 1. 1 1. 1]
 - Set **Diagram type** to **Scatter Diagram**

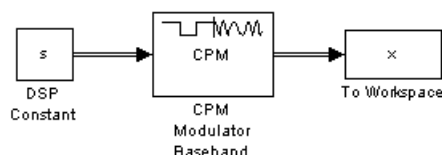
Running the model produces a scatter diagram like the one below. Your plot might look somewhat different, depending on your **Initial seed** value in the Random-Integer Generator block. Because the modulation technique is 16-QAM, the plot shows 16 clusters of points. If there were no noise, then the plot would show the 16 exact constellation points instead of clusters around the constellation points.



Phase Tree for Continuous Phase Modulation

This example plots a phase tree associated with a continuous phase modulation scheme. A phase tree is a diagram that superimposes many curves, each of which plots the phase of a modulated signal over time. The distinct curves result from different inputs to the modulator.

This example uses the CPM Modulator Baseband block for its numerical computations. The block is configured so that it uses a raised cosine filter pulse shape. The example also illustrates how you can use Simulink and MATLAB together. The example uses MATLAB commands to run a series of simulations with different input signals, to collect the simulation results, and to plot the full data set.



The first step of this example is to build the model. To open the completed model, click [here](#) in the MATLAB Help browser. To build the model, gather and configure these blocks:

- DSP Constant, in the DSP Sources library
 - Set **Constant value** to `s` (which will be in the MATLAB workspace)
 - Check the **Frame-based output** check box
- CPM Modulator Baseband
 - Set **M-ary number** to 2
 - Set **Modulation index** to $2/3$
 - Set **Frequency pulse shape** to **Raised Cosine**
 - Set **Pulse length** to 2
- To Workspace, in the Simulink Sinks library
 - Set **Variable name** to `x`
 - Set **Save format** to **Array**

Do not run the model, since the variable `s` is not yet defined in the MATLAB workspace. Instead, save the model to a directory on your MATLAB path, using the filename `doc_phasetree`.

The second step of this example is to execute these commands in MATLAB.

```

% Parameters from the CPM Modulator Baseband block
M_ary_number = 2;
modulation_index = 2/3;
pulse_length = 2;
samples_per_symbol = 8;
opts = simset('SrcWorkspace', 'Current', ...
              'DstWorkspace', 'Current');

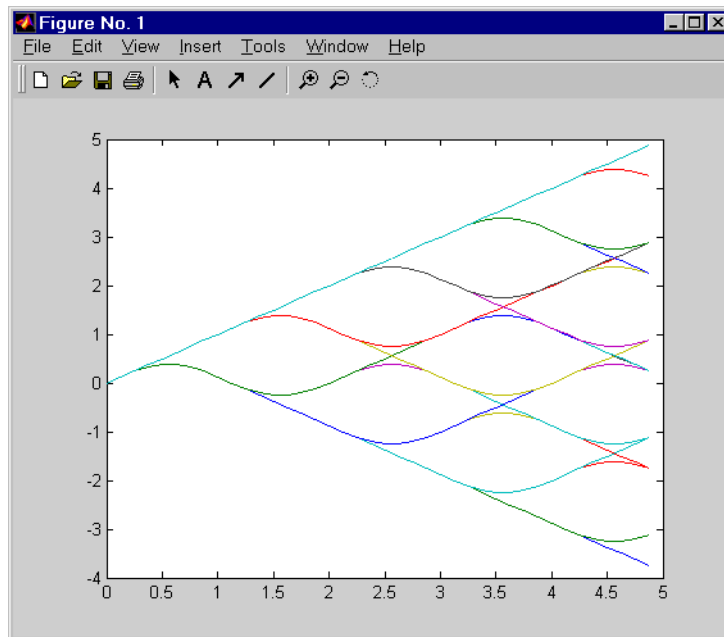
L = 5; % Symbols to display
pmat = [];
  
```

```

for ip_sig = 0: (M_ary_number^L) - 1
    s = de2bi(ip_sig, L, M_ary_number, 'left-msb');
    % Apply the mapping of the input symbol to the CPM
    % symbol 0 -> -(M-1), 1 -> -(M-2), etc.
    s = 2*s' + 1 - M_ary_number;
    sim('doc_phasetree', .9, opts); % Run model to generate x.
    pmat(:, ip_sig+1) = unwrap(angle(x(:))); % Next column of pmat
end;
pmat = pmat / (pi * modulation_index);
t = (0: L * samples_per_symbol - 1) / samples_per_symbol;
plot(t, pmat); figure(gcf); % Plot phase tree.

```

The resulting plot is below. Each curve represents a different instance of simulating the CPM Modulator Baseband block with a distinct (constant) input signal.

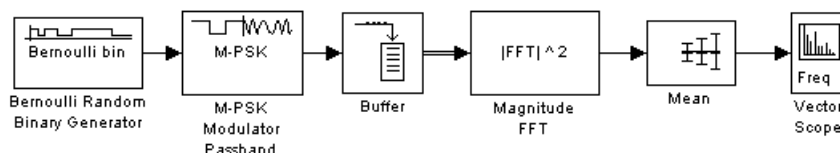


Passband Digital Modulation

The example below uses passband phase shift keying modulation and displays the spectrum of the modulated signal. The M-PSK Modulator Passband block's parameters satisfy necessary requirements for passband simulation because:

- The input signal's sampling rate of 10 is less than the carrier frequency of 100.
- The modulated signal's sampling rate of 3000 exceeds the sum of twice the carrier frequency and twice the input sampling rate.
- The input sample time of 1/10 is an integer multiple of the output sample time of 1/3000, while the modulated signal is not oversampled.

These requirements are mentioned on the reference page for the M-PSK Modulator Passband block.

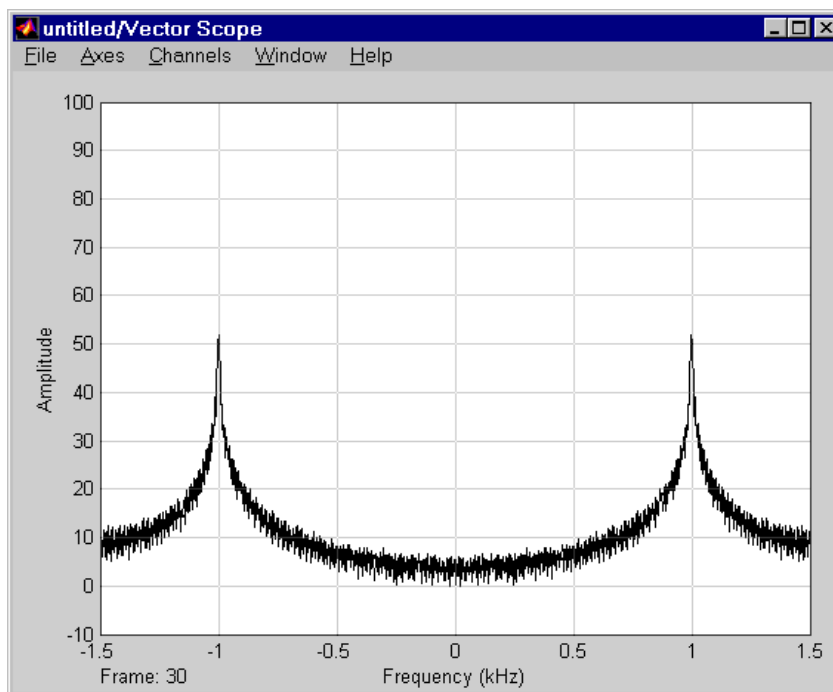


To open the completed model, click [here](#) in the MATLAB Help browser. To build the model, gather and configure these blocks:

- Bernoulli Random Binary Generator, in the Comm Sources library.
 - Set **Probability of a zero** to `[.5, .5]`
 - Set **Initial seed** to any row vector containing 2 positive integers
 - Set **Sample time** to `.1`
- M-PSK Modulator Passband, in the PM sublibrary of the Digital Passband sublibrary of Modulation
 - Set **M-ary number** to `4`
 - Set **Input type** to **Bit**
 - Set **Symbol period** to `.1`
 - Set **Carrier frequency** to `1000`
 - Set **Carrier initial phase** to `pi/4`
 - Set **Output sample time** to `1/3000`

- Buffer, in the DSP Blockset Buffers sublibrary of the Signal Management library
 - Set **Output buffer size** to 1024
- Magnitude FFT, in the DSP Blockset Power Spectrum Estimation sublibrary of the Estimation library
 - Check the **Inherit FFT length from input dimensions** check box
- Mean, in the DSP Blockset Statistics library
 - Check the **Running mean** check box
 - Set **Reset port** to **None**
- Vector Scope, in the DSP Blockset DSP Sinks library
 - Set **Input domain** to **Frequency**
 - Check the **Axis properties** check box
 - Set **Frequency range** to $[-F_s/2 \dots F_s/2]$
 - Set **Maximum Y-limit** to 100

Connect the blocks as in the figure above. Running the model produces the spectral plot below.



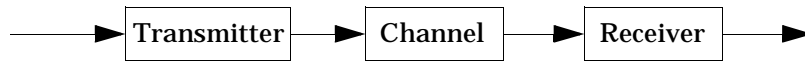
You might want to vary the modulation technique to see how this plot would change. For example, you can try replacing the M-PSK Modulator Passband block with the M-DPSK Modulator Passband or OQPSK Modulator Passband block.

Selected Bibliography for Digital Modulation

- [1] Anderson, John B., Tor Aulin, and Carl-Erik Sundberg. *Digital Phase Modulation*. New York: Plenum Press, 1986.
- [2] Jeruchim, Michel C., Philip Balaban, and K. Sam Shanmugan. *Simulation of Communication Systems*. New York: Plenum Press, 1992.
- [3] Pawula, R. F. "On M-ary DPSK Transmission Over Terrestrial and Satellite Channels." *IEEE Transactions on Communications*, vol. COM-32, July 1984. 752-761.
- [4] Smith, Joel G. "Odd-Bit Quadrature Amplitude-Shift Keying." *IEEE Transactions on Communications*, vol. COM-23, March 1975. 385-389.

Channels

Communication channels introduce noise, fading, interference, and other distortions into the signals that they transmit. Simulating a communication system involves modeling a channel based on mathematical descriptions of the channel. Different transmission media have different properties and are modeled differently. In a simulation, the channel model usually fits directly between the transmitter and receiver, as shown below.



Channel Features of the Blockset

This blockset provides several channel models for binary, real, and complex signals. You can open the Channels library by double-clicking on its icon in the main Communications Blockset library (comm1 i b), or by typing

```
commchan2
```

at the MATLAB prompt.

This section describes the capabilities of the Channels library's blocks, by considering these channels:

- Additive white Gaussian noise (AWGN) channel
- Rayleigh and Rician fading channels that model real-world mobile communication effects
- Binary symmetric channel (BSC)

AWGN Channel

An AWGN channel adds white Gaussian noise to the signal that passes through it. Gaussian noise is discussed on the reference page for the Gaussian Noise Generator block. The AWGN Channel block can process either sample-based or frame-based data, and it lets you specify the variance of the noise in one of four ways:

- Directly as a mask parameter
- Directly as an input signal

- Indirectly via a signal-to-noise ratio parameter
- Indirectly via an E_s/N_o parameter

An example using the AWGN Channel block is in “Getting Started with the Communications Blockset” on page 1-1.

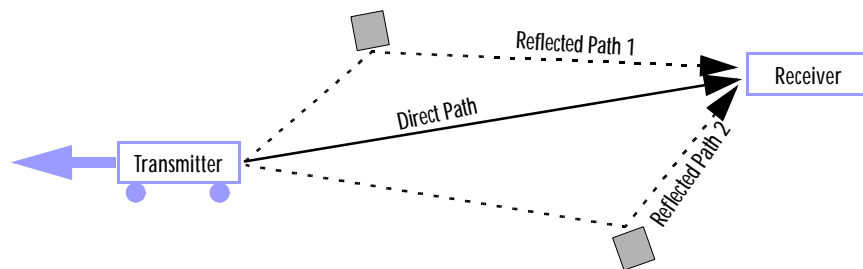
Fading Channels

The Channels library includes Rayleigh and Rician fading blocks that can simulate real-world phenomena in mobile communications. These phenomena include multipath scattering effects in the Rayleigh case, as well as Doppler shifts that arise from relative motion between the transmitter and receiver. This section discusses:

- How to categorize the possible paths along which a signal can travel from the transmitter to the receiver in the situation that you want to model
- How to choose and configure a fading channel block based on the categorization
- An example that uses fading channels

Categorizing Signal Paths

The figure below depicts the two types of paths between a moving transmitter and a stationary receiver. The solid line is a *direct line-of-sight* path, which might or might not exist in your situation. Each dotted line is a *reflected* path that the signal travels when it reflects from one of the shaded shapes. The shaded shapes represent obstacles such as buildings or trees.



The situation in the figure is just an example. In general, you should analyze your system by considering these questions:

- Are there any reflected paths along which a signal can travel from transmitter to receiver? If so, how many?
- Is there a direct path from transmitter to receiver?
- What is the *relative* motion between the transmitter and receiver?

The first two questions will help you choose which fading channel block(s) to use in your simulation, while the third question will help you choose appropriate parameters for the blocks.

Choosing and Configuring a Fading Channel Block

Once you categorize the types of signal paths in the situation you want to model, use the table below to determine the appropriate block (or blocks) for your simulation.

Table 2-5: Choosing a Fading Channel Block Based on Signal Paths

Signal Path(s)	Channel Block
Direct line-of-sight path from transmitter to receiver	Rician Fading Channel
One or more reflected paths from transmitter to receiver	Multipath Rayleigh Fading Channel

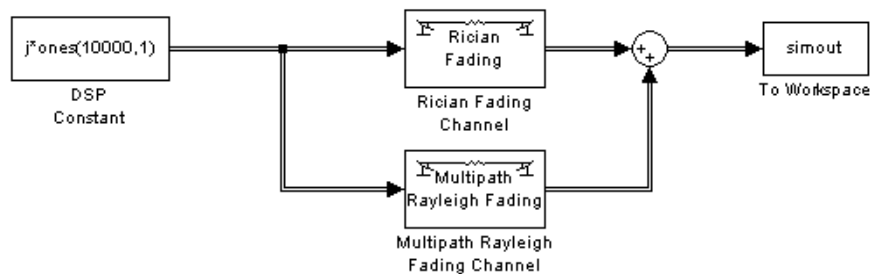
If a signal can use more than one reflected path, then a single instance of the Multipath Rayleigh Fading Channel block can model all of them simultaneously. The number of paths that the block uses is the length of either the **Delay vector** or the **Gain vector** parameter, whichever length is larger. (If both of these parameters are vectors, then they must have the same length; if exactly one of these parameters is a scalar, then the block expands it into a vector whose size matches that of the other **vector** parameter.)

The relative motion between the transmitter and receiver influences the values of the blocks' parameters. For more details, see their reference pages, as well as the works listed in "Selected Bibliography for Channels" on page 2-89 if necessary.

Example: Using Fading Channels

The reference page for the Multipath Rayleigh Fading Channel block includes an example that illustrates the channel's effect on a constant signal.

Another example is the model below, which uses both the Multipath Rayleigh Fading Channel and the Rician Fading Channel blocks in parallel. This combination of blocks simulates a mobile communication link in which the transmitted signal can travel to the receiver along a direct path as well as along three indirect paths. (The number of indirect paths is three because the Multipath Rayleigh Fading Channel block's **Gain vector** parameter is a vector of length three. Although the **Delay vector** parameter is a scalar, its value is applied to each of the three paths.)



To open the completed model, click [here](#) in the MATLAB Help browser. To build the model, gather and configure these blocks:

- DSP Constant, in the DSP Blockset DSP Sources library
 - Set **Constant value** to `j * ones(10000, 1)`
 - Check the **Frame-based output** check box
 - Set **Frame period** to `.01`
- Rician Fading Channel, with default parameter values
- Multipath Rayleigh Fading Channel
 - Set **Delay vector** to `[0 2e-6 3e-6]`
 - Set **Gain vector** to `[0 -3 1]`
- Sum, in the Simulink Math library
- To Workspace, in the Simulink Sinks directory
 - Set **Save format** to **Array**

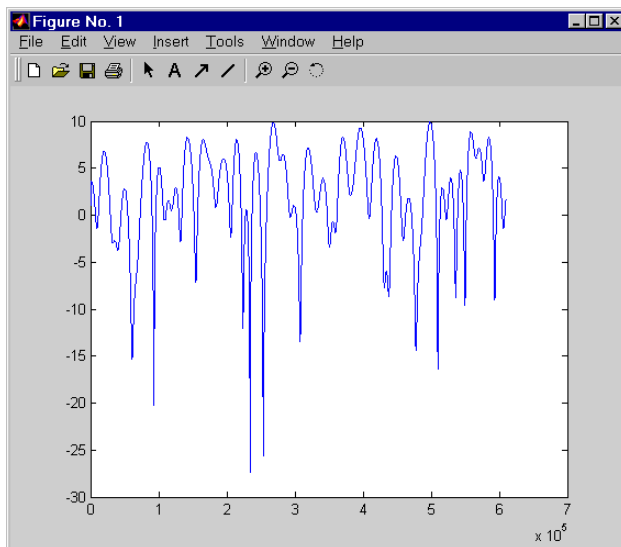
Connect the blocks as shown above. Also, from the model window's **Simulation menu**, choose **Simulation parameters**; then in the **Simulation Parameters** dialog box, set **Stop time** to `0.6`.

Tip To reduce execution time by logging less data to the workspace, set the **Decimation** parameter in the To Workspace block to 100. Then the variable `simout` will contain fewer entries, but its graph will look similar.

Run the model. After the simulation stops, plot the faded signal's power (versus sample number) by executing this command at the MATLAB prompt.

```
simout = simout.'; plot(20*log10(abs(simout(:))))
```

The resulting figure is below.



Binary Symmetric Channel

Binary error channels process binary signals by adding noise modulo 2. This library contains the Binary Symmetric Channel block, which either preserves or perturbs each vector element independently. It requires a probability that applies independently to each noise element. An example using the Binary Symmetric Channel block is in the section “Example: A Rate 2/3 Feedforward Convolutional Encoder” on page 2-42.

Selected Bibliography for Channels

- [1] Fechtel, Stefan A. "A Novel Approach to Modeling and Efficient Simulation of Frequency-Selective Fading Radio Channels." *IEEE Journal on Selected Areas in Communications*, vol. 11, April 1993. 422-431.
- [2] Jakes, William C., ed. *Microwave Mobile Communications*. New York: IEEE Press, 1974.
- [3] Lee, William C. Y. *Mobile Communications Design Fundamentals*, 2nd ed. New York: Wiley, 1993.
- [4] Proakis, John G. *Digital Communications*, 3rd ed. New York: McGraw-Hill, 1995.

Synchronization

In order to interpret information correctly, a communication receiver must be synchronized with the corresponding transmitter. A phase-locked loop, or PLL, can help accomplish this synchronization when used in conjunction with other components. A PLL is an automatic control system that adjusts the phase of a local signal to match the phase of the received signal. The PLL design works best for narrowband signals.

Synchronization Features of the Blockset

This blockset contains four phase-locked loop blocks in its Synchronization library. You can open the Synchronization library by double-clicking on its icon in the main Communications Blockset library (comml i b), or by typing

```
commsync2
```

at the MATLAB prompt.

The table below indicates which block in the Synchronization library implements each supported type of PLL.

Table 2-6: Supported PLLs in Synchronization Library

Type of PLL	Block
Analog passband PLL	Phase-Locked Loop
Analog baseband PLL	Baseband PLL
Linearized analog baseband PLL	Linearized Baseband PLL
Digital PLL using a charge pump	Charge Pump PLL

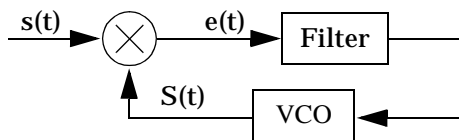
This section discusses these topics:

- “Overview of PLL Simulation” on page 2-91
- “Implementing an Analog Baseband PLL” on page 2-91
- “Implementing a Digital PLL” on page 2-92

For details about phase-locked loops, see the works listed in “Selected Bibliography for Synchronization” on page 2-92.

Overview of PLL Simulation

A simple PLL consists of a phase detector, a loop filter, and a voltage-controlled oscillator (VCO). For example, the figure below shows how these components are arranged for an analog passband PLL. In this case, the phase detector is just a multiplier. The signal $e(t)$ is often called the error signal.



Different PLLs use different phase detectors, filters, and VCO characteristics. Some of these attributes are built into the PLL blocks in this blockset, while others depend on parameters that you set in the block mask:

- You specify the filter's transfer function in the block mask using the **Lowpass filter numerator** and **Lowpass filter denominator** parameters. Each of these parameters is a vector that lists the coefficients of the respective polynomial in order of descending exponents of the variable s . To design a filter, you can use functions such as `butter`, `cheby1`, and `cheby2` in the Signal Processing Toolbox.
- You specify the key VCO characteristics in the block mask. All four PLL blocks use a **VCO input sensitivity** parameter. Some blocks also use **VCO quiescent frequency**, **VCO initial phase**, and **VCO output amplitude** parameters.
- The phase detector for each of the PLL blocks is a feature that you cannot change from the block mask.

Implementing an Analog Baseband PLL

Unlike passband models for a phase-locked loop, a baseband model does not depend on a carrier frequency. This allows you to use a lower sampling rate in the simulation. These two blocks implement analog baseband PLLs:

- Baseband PLL
- Linearized Baseband PLL

The linearized model and the nonlinearized model differ in that the linearized model uses the approximation

$$\sin(\Delta\theta(t)) \cong \Delta\theta(t)$$

to simplify the computations. This approximation is close when $\Delta\theta(t)$ is near zero. Thus, instead of using the input signal and the VCO output signal directly, the linearized PLL model uses only their *phases*.

Implementing a Digital PLL

The charge pump PLL is a classical digital PLL. Unlike the analog PLLs mentioned above, the charge pump PLL uses a sequential logic phase detector, which is also known as a digital phase detector or a phase/frequency detector.

Selected Bibliography for Synchronization

- [1] Gardner, F. M. "Charge-pump Phase-lock Loops." *IEEE Trans. on Communications*, vol. 28, November 1980. 1849-1858.
- [2] Gardner, F. M. "Phase Accuracy of Charge Pump PLLs." *IEEE Trans. on Communications*, vol. 30, October 1982. 2362-2363.
- [3] Gupta, S. C. "Phase Locked Loops." *Proceedings of the IEEE*, vol. 63, February 1975. 291-306.
- [4] Lindsay, W. C. and C. M. Chie. "A Survey on Digital Phase-Locked Loops." *Proceedings of the IEEE*, vol. 69, April 1981. 410-431.
- [5] Meyr, Heinrich and Ascheid, Gerd. *Synchronization in Digital Communications*, vol. 1. New York: John Wiley & Sons, 1990.

Function Reference

Alphabetical List of Functions

<code>comm_links</code>	3-3
<code>commlib</code>	3-4

Purpose	Display library link information for Communications Blockset blocks
Syntax	<code>comm_links</code> <code>comm_links(sys)</code> <code>comm_links(sys, mode)</code>
Description	<p><code>comm_links</code> displays library link information for blocks in the current model that are linked to the Communications Blockset. For each block in the current model, <code>comm_links</code> replaces the block name with the full pathname to the block's library link in the Communications Blockset. Blocks linked to the current Communications Blockset libraries are highlighted in blue. Blocks linked to older versions of the Simulink portion of the Communications Toolbox are highlighted in red. Blocks at all levels of the model are analyzed.</p> <p>A summary report indicating the number of blocks linked to each blockset version is also displayed in the MATLAB command window. The highlighting and link display are disabled when the model is executed or saved, or when <code>comm_links</code> is executed a second time from the MATLAB command line.</p> <p><code>comm_links(sys)</code> toggles the display of block links in system <code>sys</code>. If <code>sys</code> is the current model (<code>gcs</code>), this is the same as the earlier <code>comm_links</code> syntax.</p> <p><code>comm_links(sys, mode)</code> directly sets the link display state, where <code>mode</code> can be 'on', 'off', or 'toggle'. The default is 'toggle'.</p>
See Also	<code>liblinks</code> (DSP Blockset)

commlib

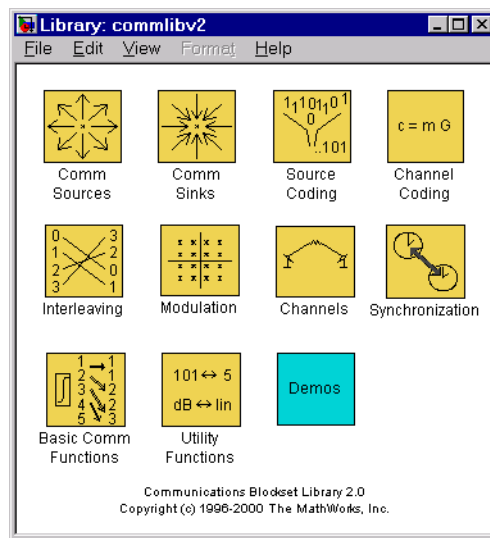
Purpose	Open the main Communications Blockset library
Syntax	<code>commlib</code> <code>commlib(n)</code> <code>commlib n</code>
Description	<p><code>commlib</code> opens the current version of the main Communications Blockset library.</p> <p><code>commlib(n)</code> opens version number <code>n</code> of the main Communications Blockset library, where <code>n</code> can be either 1.3, 1.5, or 2.0. Version 2.0 refers to the Release 12 Communications Blockset. Version 1.5 refers to the Simulink portion of the Communications Toolbox 1.5 (Release 11.1). Version 1.3 refers to the Simulink portion of the Communications Toolbox 1.3 (Release 10).</p> <p><code>commlib n</code> is the same as <code>commlib(n)</code>.</p>
See Also	<code>simulink3</code> (Simulink), <code>dsplib</code> (DSP Blockset)

Block Reference

Communications Sources	4-3
Communications Sinks	4-5
Source Coding	4-7
Channel Coding	4-9
Block Coding	4-9
Convolutional Coding	4-11
Interleaving	4-13
Block Interleaving	4-13
Convolutional Interleaving	4-15
Modulation	4-18
Digital Baseband Modulation	4-18
Analog Baseband Modulation	4-24
Digital Passband Modulation	4-26
Analog Passband Modulation	4-31
Channels	4-34
Synchronization	4-36
Basic Communications Functions	4-37
Integrators	4-37
Sequence Operations	4-38
Utility Functions	4-41

This chapter contains detailed descriptions of all Communications Blockset blocks. It first shows the libraries and lists their contents, and then presents the block reference entries in alphabetical order. More detailed discussions of the core libraries' capabilities are in the previous chapter.

Below is the Communications Blockset. You can open it by typing `commlib` at the MATLAB prompt. Each yellow icon in this window represents a library. In Simulink, double-clicking on a library icon opens the library. In this document, clicking on one of the yellow icons below jumps to an overview of that library.



To access an older version of the library (for example, if you are modifying one of your legacy models), then you should use one of these alternative syntaxes of the `commlib` command.

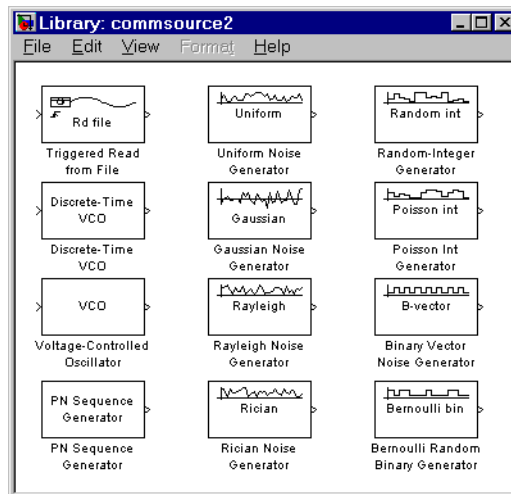
```
commlib 1.3 % To open version 1.3
```

```
commlib 1.5 % To open version 1.5
```

Communications Sources

Every communication system contains one or more sources. You can open the Comm Sources library by double-clicking on its icon in the main Communications Blockset library (commlib), or by typing `commsource2` at the MATLAB prompt.

Tip In this document, you can jump to a block's reference page by clicking on its icon below.



The table below lists and describes the blocks in the Comm Sources library. For information about a specific block, see the reference pages that follow; for a

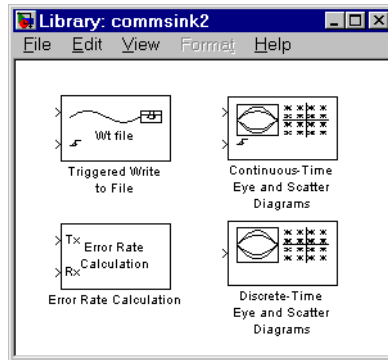
discussion of this library’s capabilities, see “Communications Sources” on page 2-6.

Block Name	Purpose
Bernoulli Random Binary Generator	Generate Bernoulli-distributed random binary numbers
Binary Vector Noise Generator	Generate a binary vector while controlling the number of 1s
Discrete-Time VCO	Implement a voltage-controlled oscillator in discrete time
Gaussian Noise Generator	Generate Gaussian distributed noise with given mean and variance values
PN Sequence Generator	Generate pseudonoise sequence
Poisson Int Generator	Generate Poisson-distributed random integers
Random-Integer Generator	Generate integers randomly distributed in the range [0, M-1]
Rayleigh Noise Generator	Generate Rayleigh distributed noise
Rician Noise Generator	Generate Rician distributed noise
Triggered Read From File	Read from a file, refreshing the output at rising edges of an input signal
Uniform Noise Generator	Generate uniformly distributed noise between the upper and lower bounds
Voltage-Controlled Oscillator	Implement a voltage-controlled oscillator

Communications Sinks

The Comm Sinks library provides sinks and display devices that facilitate analysis of communication system performance. You can open the Comm Sinks library by double-clicking on its icon in the main Communications Blockset library (commlib), or by typing `commsink2` at the MATLAB prompt.

Tip In this document, you can jump to a block's reference page by clicking on its icon below.



The table below lists and describes the blocks in the Comm Sinks library. For information about a specific block, see the reference pages that follow; for a discussion of this library's capabilities, see "Communications Sinks" on page 2-12.

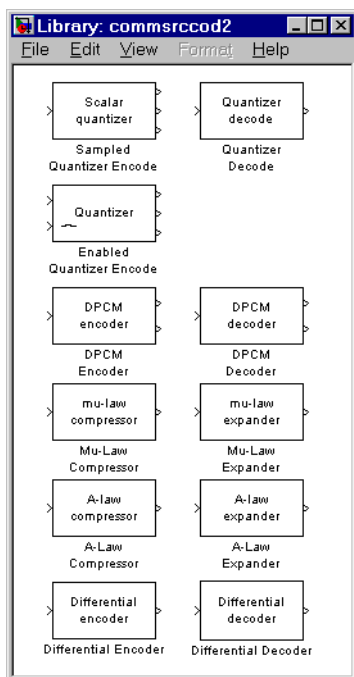
Block Name	Purpose
Error Rate Calculation	Compute the bit error rate or symbol error rate of input data
Continuous-Time Eye and Scatter Diagrams	Produce eye diagram, scatter, or x-y plots, using trigger to set decision timing

Block Name (Continued)	Purpose (Continued)
Discrete-Time Eye and Scatter Diagrams	Produce an eye diagram and/or scatter diagram
Triggered Write to File	Write to a file at each rising edge of an input signal

Source Coding

This blockset supports companders, scalar quantization and predictive quantization. You can open the Source Coding library by double-clicking on its icon in the main Communications Blockset library (commlib), or by typing `commsrccod2` at the MATLAB prompt.

Tip In this document, you can jump to a block's reference page by clicking on its icon below.



The table below lists and describes the blocks in the Source Coding library. For information about a specific block, see the reference pages that follow; for a discussion of this library’s capabilities, see “Source Coding” on page 2-16.

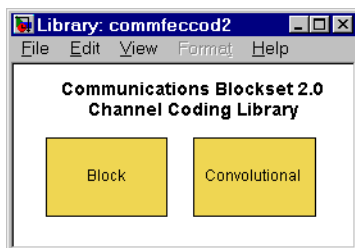
Block Name	Purpose
A-Law Compressor	Implement A-law compressor for source coding
A-Law Expander	Implement A-law expander for source coding
Differential Decoder	Decode a binary signal using differential coding technique.
Differential Encoder	Encode a binary signal using differential coding technique.
DPCM Decoder	Decode differential pulse code modulation
DPCM Encoder	Encode using differential pulse code modulation
Mu-Law Compressor	Implement m-law compressor for source coding
Mu-Law Expander	Implement m-law expander for source coding
Quantizer Decode	Decode quantization index according to codebook
Sampled Quantizer Encode	Quantize a signal, indicating quantization index, coded signal, and distortion
Enabled Quantizer Encode	Quantize a signal, using trigger to control processing

Channel Coding

The Channel Coding library contains two sublibraries:

- Block, which contains blocks that implement the encoding and decoding of linear, cyclic, BCH, Hamming, and Reed-Solomon codes
- Convolutional, which contains blocks that implement convolutional encoding and decoding

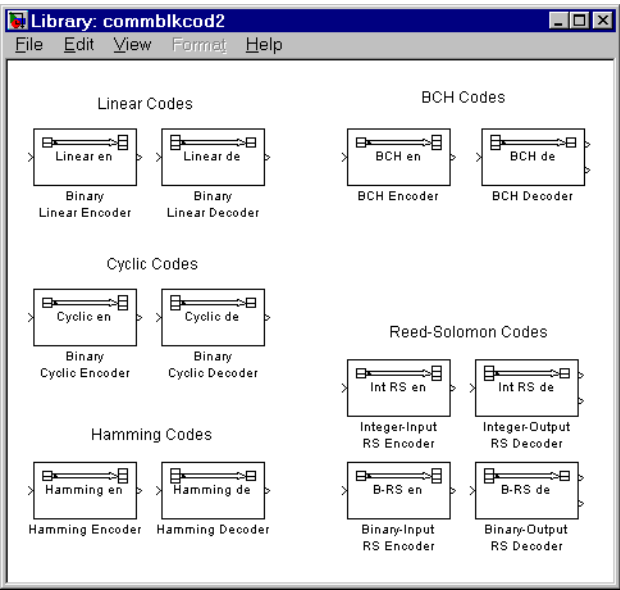
The main Channel Coding library appears below. You can open it by double-clicking on its icon in the main Communications Blockset library (`commlib`), or by typing `commfeccod2` at the MATLAB prompt. Each icon in the Channel Coding window represents a sublibrary. In Simulink, double-clicking on one of these icons opens the sublibrary. In this document, clicking on one of the icons below jumps to an overview of that sublibrary.



Block Coding

You can open the Block sublibrary by double-clicking on the Block icon in the main Channel Coding library, or by typing `commblkcod2` at the MATLAB prompt.

Tip In this document, you can jump to a block's reference page by clicking on its icon below.



The table below lists and describes the blocks in the Block sublibrary of the Channel Coding library. For information about a specific block, see the reference pages that follow; for a discussion of this library’s capabilities, see “Block Coding” on page 2-27.

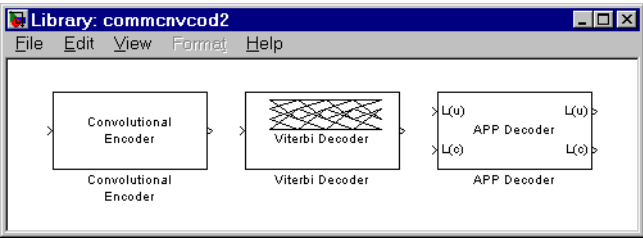
Block Name	Purpose
BCH Decoder	Decode a BCH code to recover binary vector data
BCH Encoder	Create a BCH code from binary vector data
Binary Cyclic Decoder	Decode a systematic cyclic code to recover binary vector data
Binary Cyclic Encoder	Create a systematic cyclic code from binary vector data

Block Name (Continued)	Purpose (Continued)
Binary-Output RS Decoder	Decode a Reed-Solomon code to recover binary vector data
Binary-Input RS Encoder	Create a Reed-Solomon code from binary vector data
Binary Linear Decoder	Decode a linear block code to recover binary vector data
Binary Linear Encoder	Create a linear block code from binary vector data
Hamming Decoder	Decode a Hamming code to recover binary vector data
Hamming Encoder	Create a Hamming code from binary vector data
Integer-Output RS Decoder	Decode a Reed-Solomon code to recover integer vector data
Integer-Input RS Encoder	Create a Reed-Solomon code from integer vector data

Convolutional Coding

You can open the Convolutional sublibrary by double-clicking on the Convolutional icon in the main Channel Coding library, or by typing `commcnvcod2` at the MATLAB prompt.

Tip In this document, you can jump to a block's reference page by clicking on its icon below.



The table below lists and describes the blocks in the Convolutional sublibrary of the Channel Coding library. For information about a specific block, see the reference pages that follow; for a discussion of this library’s capabilities, see “Convolutional Coding” on page 2-40.

Block Name	Purpose
APP Decoder	Decode a convolutional code using the a posteriori probability (APP) method
Convolutional Encoder	Create a convolutional code from binary data
Viterbi Decoder	Decode convolutionally encoded data using the Viterbi algorithm

Interleaving

The Interleaving library contains two sublibraries:

- Block
- Convolutional

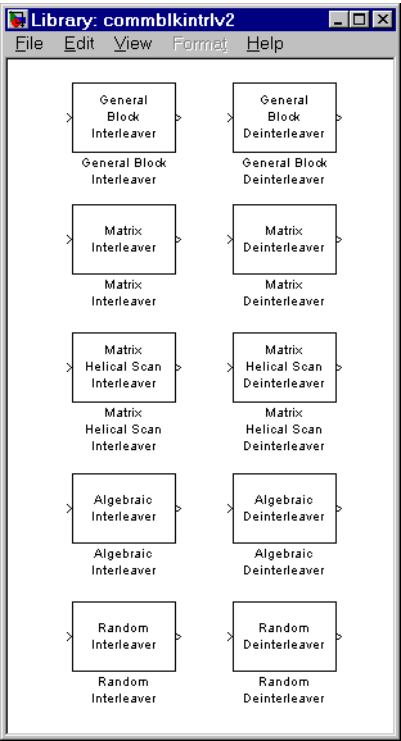
The main Interleaving library appears below. You can open it by double-clicking on its icon in the main Communications Blockset library (commlib), or by typing `comminterleave2` at the MATLAB prompt. Each icon in the Interleaving window represents a sublibrary. In Simulink, double-clicking on one of these icons opens the sublibrary. In this document, clicking on one of the icons below jumps to an overview of that sublibrary.



Block Interleaving

You can open the Block sublibrary by double-clicking on the Block icon in the main Interleaving library, or by typing `commblkinterlv2` at the MATLAB prompt.

Tip In this document, you can jump to a block's reference page by clicking on its icon below.



The table below lists and describes the blocks in the Block sublibrary of the Interleaving library. For information about a specific block, see the reference pages that follow; for a discussion of this library’s capabilities, see “Block Interleavers” on page 2-46.

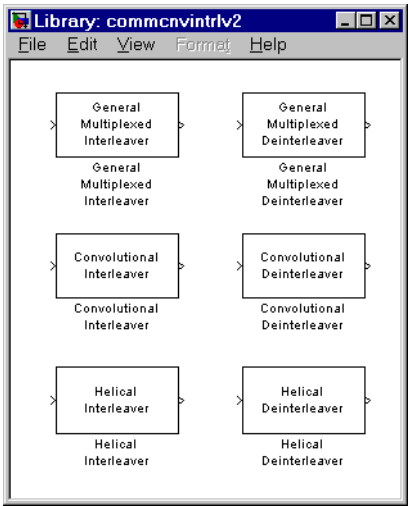
Block Name	Purpose
Algebraic Deinterleaver	Restore ordering of the input symbols using algebraically derived permutation
Algebraic Interleaver	Reorder the input symbols using algebraically derived permutation table

Block Name (Continued)	Purpose (Continued)
General Block Deinterleaver	Restore ordering of the symbols in the input vector
General Block Interleaver	Reorder the symbols in the input vector
Matrix Deinterleaver	Permute input symbols by filling a matrix by columns and emptying it by rows
Matrix Helical Scan Deinterleaver	Restore ordering of input symbols by filling a matrix along diagonals
Matrix Helical Scan Interleaver	Permute input symbols by selecting matrix elements along diagonals
Matrix Interleaver	Permute input symbols by filling a matrix by rows and emptying it by columns
Random Deinterleaver	Restore ordering of the input symbols using a random permutation
Random Interleaver	Reorder the input symbols using a random permutation

Convolutional Interleaving

You can open the Convolutional sublibrary by double-clicking on the Convolutional icon in the main Interleaving library, or by typing `commcnvntrl v2` at the MATLAB prompt.

Tip In this document, you can jump to a block’s reference page by clicking on its icon below.



The table below lists and describes the blocks in the Convolutional sublibrary of the Interleaving library. For information about a specific block, see the reference pages that follow; for a discussion of this library’s capabilities, see “Convolutional Interleavers” on page 2-47.

Block Name	Purpose
Convolutional Deinterleaver	Restore ordering of symbols that were permuted using shift registers
Convolutional Interleaver	Permute input symbols using a set of shift registers
General Multiplexed Deinterleaver	Restore ordering of symbols using specified-delay shift registers
General Multiplexed Interleaver	Permute input symbols using a set of shift registers with specified delays

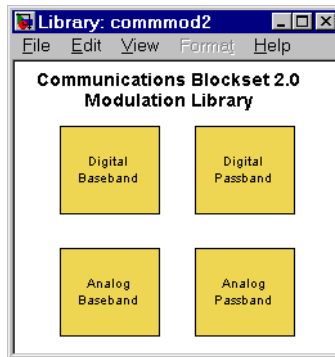
Block Name (Continued)	Purpose (Continued)
Helical Deinterleaver	Restore ordering of symbols permuted by a helical interleaver
Helical Interleaver	Permute input symbols using a helical array

Modulation

The Modulation library contains four sublibraries, each of which addresses a category of modulation:

- Digital Baseband Modulation
- Analog Baseband Modulation
- Digital Passband Modulation
- Analog Passband Modulation

The main Modulation library appears below. You can open it by double-clicking on its icon in the main Communications Blockset library (comm1 i b), or by typing `commmod2` at the MATLAB prompt. Each icon in the Modulation window represents a sublibrary. In Simulink, double-clicking on one of these icons opens the sublibrary. In this document, clicking on one of the icons below jumps to an overview of that sublibrary.

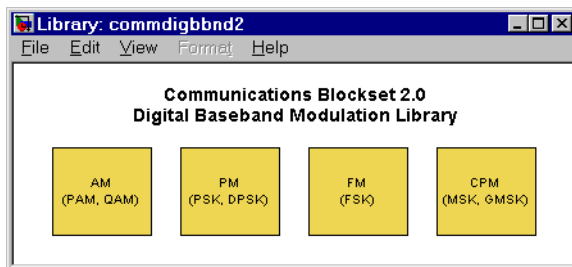


The first column shows the sublibraries for baseband simulation; the second column shows the sublibraries for passband simulation. The first row shows the sublibraries for digital modulation and demodulation. The second row shows the sublibraries for analog modulation and demodulation.

Digital Baseband Modulation

You can open the Digital Baseband sublibrary of Modulation by double-clicking on the Digital Baseband icon in the main Modulation library, or by typing

`commdi gbbnd2` at the MATLAB prompt. clicking on one of the icons below jumps to an overview of that sublibrary.

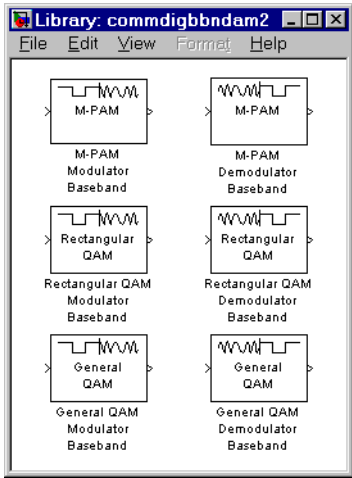


Digital Baseband is further divided into sublibraries according to specific modulation techniques:

- Amplitude modulation (PAM, QAM)
- Phase modulation (PSK, DPSK)
- Frequency modulation (FSK)
- Continuous phase modulation (MSK, GMSK)

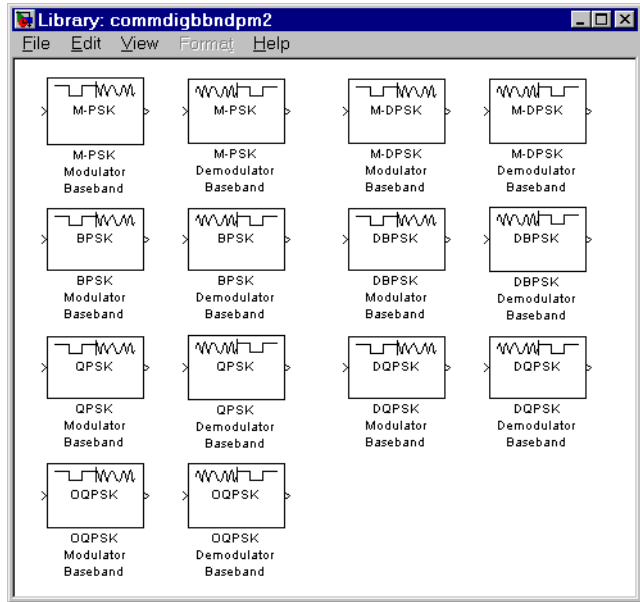
The figures and tables below show and list the blocks in the method-specific sublibraries. For information about a specific block, see the reference pages that follow; for a discussion of digital baseband modulation capabilities, see “Digital Modulation” on page 2-64.

AM Sublibrary



Block Name	Purpose
General QAM Demodulator Baseband	Demodulate QAM-modulated data
General QAM Modulator Baseband	Modulate using quadrature amplitude modulation
M-PAM Demodulator Baseband	Demodulate PAM-modulated data
M-PAM Modulator Baseband	Modulate using M-ary pulse amplitude modulation
Rectangular QAM Demodulator Baseband	Demodulate QAM-modulated data
Rectangular QAM Modulator Baseband	Modulate using M-ary quadrature amplitude modulation

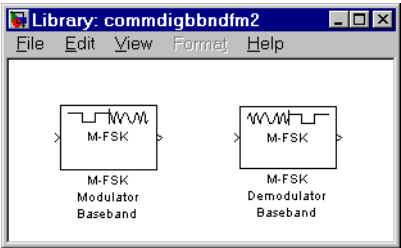
PM Sublibrary



Block Name	Purpose
BPSK Demodulator Baseband	Demodulate BPSK-modulated data
BPSK Modulator Baseband	Modulate using the binary phase shift keying method
DBPSK Demodulator Baseband	Demodulate DBPSK-modulated data
DBPSK Modulator Baseband	Modulate using the differential binary phase shift keying method
DQPSK Demodulator Baseband	Demodulate DQPSK-modulated data
DQPSK Modulator Baseband	Modulate using the differential quaternary phase shift keying method

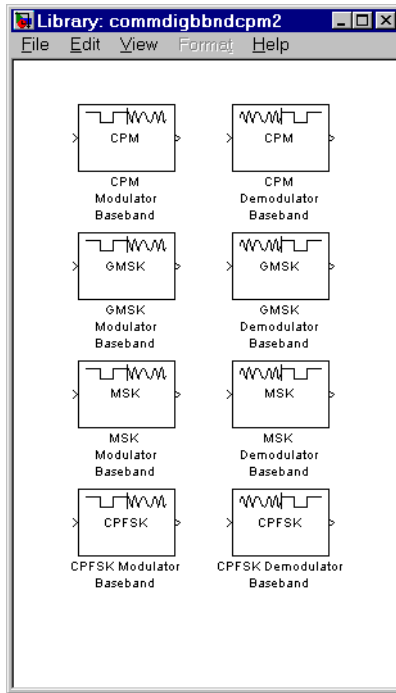
Block Name (Continued)	Purpose (Continued)
M-DPSK Demodulator Baseband	Demodulate DPSK-modulated data
M-DPSK Modulator Baseband	Modulate using the M-ary differential phase shift keying method
M-PSK Demodulator Baseband	Demodulate PSK-modulated data
M-PSK Modulator Baseband	Modulate using the M-ary phase shift keying method
OQPSK Demodulator Baseband	Demodulate OQPSK-modulated data
OQPSK Modulator Baseband	Modulate using the offset quadrature phase shift keying method
QPSK Demodulator Baseband	Demodulate QPSK-modulated data
QPSK Modulator Baseband	Modulate using the quaternary phase shift keying method

FM Sublibrary



Block Name	Purpose
M-FSK Demodulator Baseband	Demodulate FSK-modulated data
M-FSK Modulator Baseband	Modulate using the M-ary frequency shift keying method

CPM Sublibrary



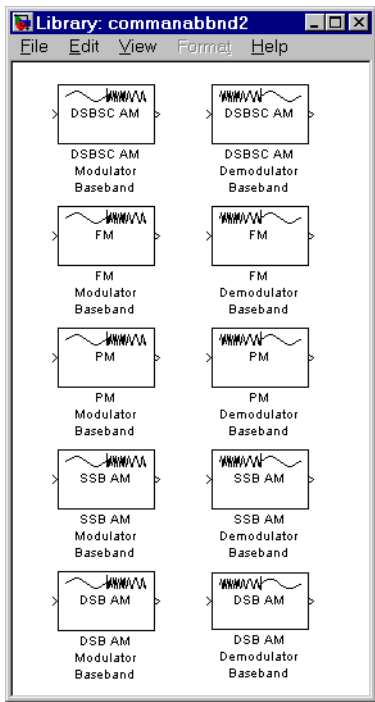
Block Name	Purpose
CPFSK Demodulator Baseband	Demodulate CPFSK-modulated data
CPFSK Modulator Baseband	Modulate using the continuous phase frequency shift keying method
CPM Demodulator Baseband	Demodulate CPM-modulated data
CPM Modulator Baseband	Modulate using continuous phase modulation
GMSK Demodulator Baseband	Demodulate GMSK-modulated data

Block Name (Continued)	Purpose (Continued)
GMSK Modulator Baseband	Modulate using the Gaussian minimum shift keying method
MSK Demodulator Baseband	Demodulate MSK-modulated data
MSK Modulator Baseband	Modulate using the minimum shift keying method

Analog Baseband Modulation

You can open the Analog Baseband sublibrary of Modulation by double-clicking on the Analog Baseband icon in the main Modulation library, or by typing `commanabbnd2` at the MATLAB prompt.

Tip In this document, you can jump to a block's reference page by clicking on its icon below.



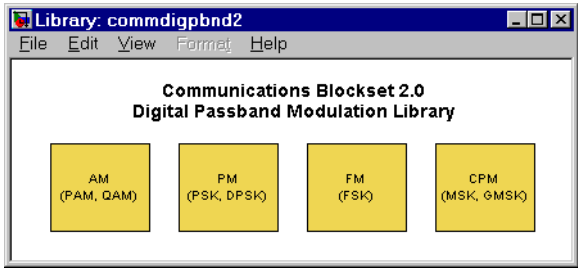
The table below lists and describes the blocks in the Analog Baseband sublibrary of the Modulation library. For information about a specific block, see the reference pages that follow; for a discussion of this library’s capabilities, see “Analog Modulation” on page 2-53.

Block Name	Purpose
DSB AM Demodulator Baseband	Demodulate DSB-AM-modulated data
DSB AM Modulator Baseband	Modulate using double-sideband amplitude modulation
DSBSC AM Demodulator Baseband	Demodulate DSBSC-AM-modulated data

Block Name (Continued)	Purpose (Continued)
DSBSC AM Modulator Baseband	Modulate using double-sideband suppressed-carrier amplitude modulation
FM Demodulator Baseband	Demodulate FM-modulated data
FM Modulator Baseband	Modulate using frequency modulation
PM Demodulator Baseband	Demodulate PM-modulated data
PM Modulator Baseband	Modulate using phase modulation
SSB AM Demodulator Baseband	Demodulate SSB-AM-modulated data
SSB AM Modulator Baseband	Modulate using single-sideband amplitude modulation

Digital Passband Modulation

You can open the Digital Passband sublibrary of Modulation by double-clicking on the Digital Passband icon in the main Modulation library, or by typing `commdiagpbnd2` at the MATLAB prompt. clicking on one of the icons below jumps to an overview of that sublibrary.



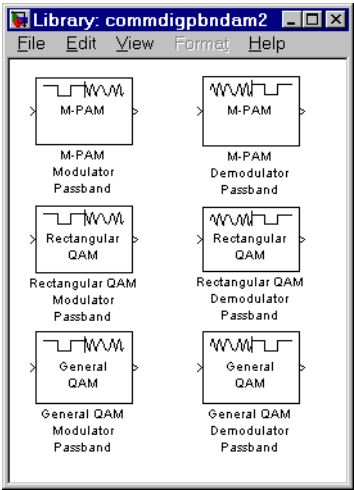
Digital Passband is further divided into sublibraries according to specific modulation techniques:

- Amplitude modulation (PAM, QAM)
- Phase modulation (PSK, DPSK)

- Frequency modulation (FSK)
- Continuous phase modulation (MSK, GMSK)

The figures and tables below show and list the blocks in the method-specific sublibraries. For information about a specific block, see the reference pages that follow.

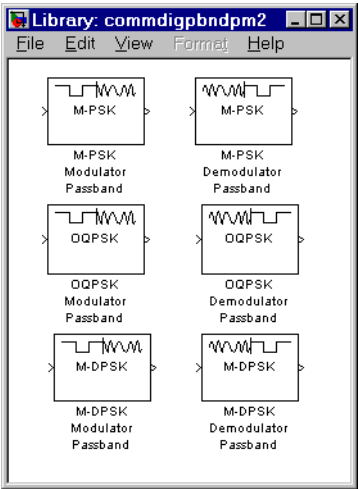
AM Sublibrary



Block Name	Purpose
General QAM Demodulator Passband	Demodulate QAM-modulated data
General QAM Modulator Passband	Modulate using the pulse amplitude modulation phase shift keying method
M-PAM Demodulator Passband	Demodulate PAM-modulated data
M-PAM Modulator Passband	Modulate using M-ary pulse amplitude modulation

Block Name (Continued)	Purpose (Continued)
Rectangular QAM Demodulator Passband	Demodulate QAM-modulated data
Rectangular QAM Modulator Passband	Modulate using M-ary quadrature amplitude modulation

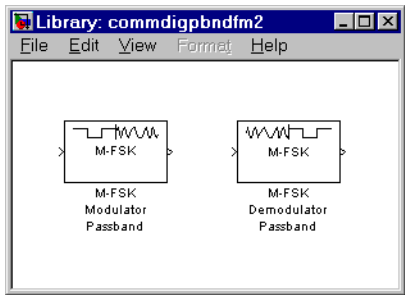
PM Sublibrary



Block Name	Purpose
M-DPSK Demodulator Passband	Demodulate DPSK-modulated data
M-DPSK Modulator Passband	Modulate using the M-ary differential phase shift keying method
M-PSK Demodulator Passband	Demodulate PSK-modulated data
M-PSK Modulator Passband	Modulate using the M-ary phase shift keying method

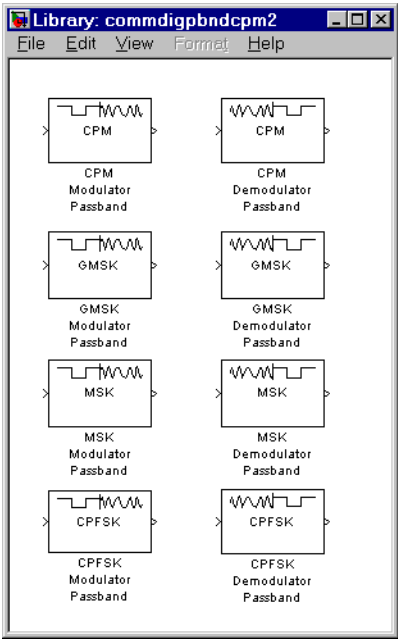
Block Name (Continued)	Purpose (Continued)
OQPSK Demodulator Passband	Demodulate OQPSK-modulated data
OQPSK Modulator Passband	Modulate using the offset quadrature phase shift keying method

FM Sublibrary



Block Name	Purpose
M-FSK Demodulator Passband	Modulate using the M-ary frequency shift keying method
M-FSK Modulator Passband	Modulate using the M-ary frequency shift keying method

CPM Sublibrary



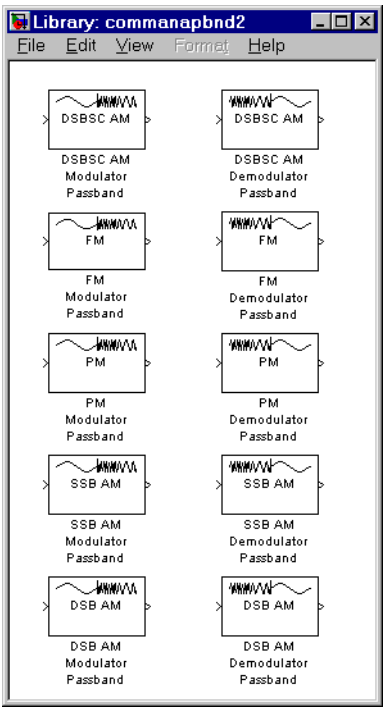
Block Name	Purpose
CPFSK Demodulator Passband	Demodulate CPFSK-modulated data
CPFSK Modulator Passband	Modulate using the continuous phase frequency shift keying method
CPM Demodulator Passband	Demodulate CPM-modulated data
CPM Modulator Passband	Modulate using continuous phase modulation
GMSK Demodulator Passband	Demodulate GMSK-modulated data
GMSK Modulator Passband	Modulate using the Gaussian minimum shift keying method

Block Name (Continued)	Purpose (Continued)
MSK Demodulator Passband	Demodulate MSK-modulated data
MSK Modulator Passband	Modulate using the minimum shift keying method

Analog Passband Modulation

You can open the Analog Passband sublibrary of Modulation by double-clicking on the Analog Passband icon in the main Modulation library, or by typing `commmanapbnd2` at the MATLAB prompt.

Tip In this document, you can jump to a block's reference page by clicking on its icon below.



The table below lists and describes the blocks in the Analog Passband sublibrary of the Modulation library. For information about a specific block, see the reference pages that follow; for a discussion of this library’s capabilities, see “Analog Modulation” on page 2-53.

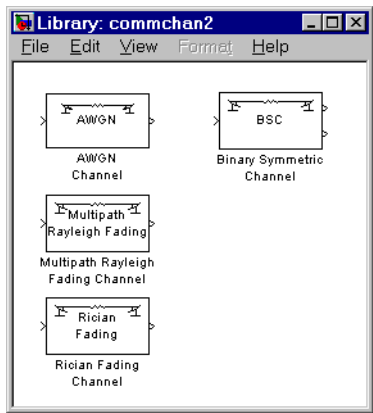
Block Name	Purpose
DSB AM Demodulator Passband	Demodulate DSB-AM-modulated data
DSB AM Modulator Passband	Modulate using double-sideband amplitude modulation
DSBSC AM Demodulator Passband	Demodulate DSBSC-AM-modulated data

Block Name (Continued)	Purpose (Continued)
DSBSC AM Modulator Passband	Modulate using double-sideband suppressed-carrier amplitude modulation
FM Demodulator Passband	Demodulate FM-modulated data
FM Modulator Passband	Modulate using frequency modulation
PM Demodulator Passband	Demodulate PM-modulated data
PM Modulator Passband	Modulate using phase modulation
SSB AM Demodulator Passband	Demodulate SSB-AM-modulated data
SSB AM Modulator Passband	Modulate using single-sideband amplitude modulation

Channels

The Channels library provides passband and baseband channels. You can open the Channels library by double-clicking on its icon in the main Communications Blockset library (comm lib), or by typing `commchan2` at the MATLAB prompt.

Tip In this document, you can jump to a block’s reference page by clicking on its icon below.



The table below lists and describes the blocks in the Channels library. For information about a specific block, see the reference pages that follow; for a discussion of this library’s capabilities, see “Channels” on page 2-84.

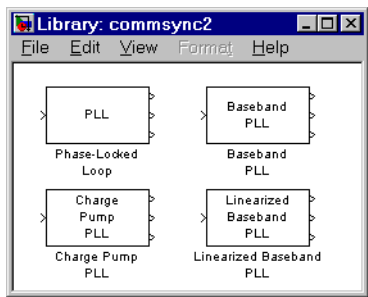
Block Name	Purpose
AWGN Channel	Add white Gaussian noise to the input signal
Binary Symmetric Channel	Introduce binary errors

Block Name (Continued)	Purpose (Continued)
Multipath Rayleigh Fading Channel	Simulate a multipath Rayleigh fading propagation channel
Rician Fading Channel	Simulate a Rician fading propagation channel

Synchronization

The Synchronization library provides four phase-locked loop models. You can open the Synchronization library by double-clicking on its icon in the main Communications Blockset library (commlib), or by typing `commsync2` at the MATLAB prompt.

Tip In this document, you can jump to a block’s reference page by clicking on its icon below.



The table below lists and describes the blocks in the Synchronization library. For information about a specific block, see the reference pages that follow; for a discussion of this library’s capabilities, see “Synchronization” on page 2-90.

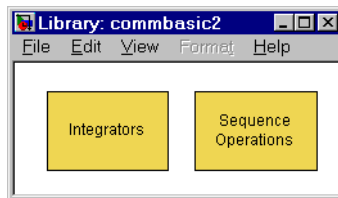
Block Name	Purpose
Baseband PLL	Implement a baseband phase-locked loop
Charge Pump PLL	Implement a charge pump phase-locked loop using a digital phase detector
Linearized Baseband PLL	Implement a linearized version of a baseband phase-locked loop
Phase-Locked Loop	Implement a phase-locked loop to recover the phase of the input signal

Basic Communications Functions

The Basic Comm Functions library contains these sublibraries:

- Integrators
- Sequence Operations

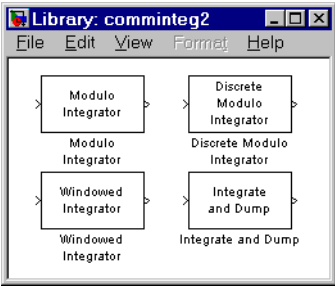
The main Basic Comm Functions library appears below. You can open it by double-clicking on its icon in the main Communications Blockset library (comm lib), or by typing `commbasic2` at the MATLAB prompt. Each icon in the Basic Comm Functions window represents a sublibrary. In Simulink, double-clicking on one of these icons opens the sublibrary. In this document, clicking on one of the icons below jumps to an overview of that sublibrary.



Integrators

You can open the Integrators sublibrary by double-clicking on the Integrators icon in the main Basic Comm Functions library, or by typing `comminteg2` at the MATLAB prompt.

Tip In this document, you can jump to a block's reference page by clicking on its icon below.



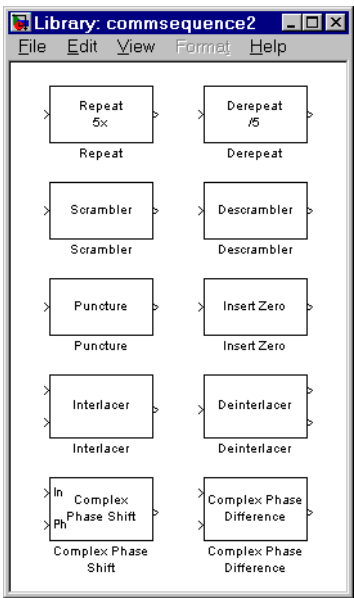
The table below lists and describes the blocks in the Integrators library. For information about a specific block, see the reference pages that follow.

Block Name	Purpose
Discrete Modulo Integrator	Integrate in discrete time and reduce by a modulus
Integrate and Dump	Integrate, resetting to zero periodically and reducing by a modulus
Modulo Integrator	Integrate in continuous time and reduce by a modulus
Windowed Integrator	Integrate over a time window of fixed length

Sequence Operations

You can open the Sequence Operations sublibrary by double-clicking on the Sequence Operations icon in the main Basic Comm Functions library, or by typing `commsequence2` at the MATLAB prompt.

Tip In this document, you can jump to a Communications Blockset block's reference page by clicking on its icon below. (The Repeat block is in the DSP Blockset, not the Communications Blockset; the icon in the Sequence Operations sublibrary is merely a link to the DSP Blockset.)



The table below lists and describes the blocks in the Sequence Operations library. For information about a specific block, see the reference pages that follow.

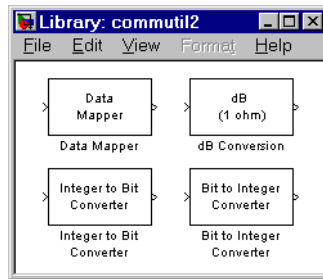
Block Name	Purpose
Complex Phase Difference	Output the phase difference between the two complex input signals
Complex Phase Shift	Shift the phase of the complex input signal by the second input value
Deinterlacer	Distribute elements of input vector alternately between two output vectors
Derepeat	Reduce sampling rate by averaging consecutive samples
Descrambler	Descramble the input signal

Block Name	Purpose (Continued)
Insert Zero	Distribute input elements in output vector
Interlacer	Alternately select elements from two input vectors to generate output vector
Puncture	Output the elements which correspond to 1s in the binary Puncture vector
Repeat	Repeat input samples N times
Scrambler	Scramble the input signal

Utility Functions

You can open the Utility Functions library by double-clicking on its icon in the main Communications Blockset library (`comm11b`), or by typing `commutil2` at the MATLAB prompt.

Tip In this document, you can jump to a Communications Blockset block's reference page by clicking on its icon below. (The dB Conversion block is in the DSP Blockset, not the Communications Blockset; the icon in the Utility Functions library is merely a link to the DSP Blockset.)



The table below lists and describes the blocks in the Utility Functions library. For information about a specific block, see the reference pages that follow.

Block Name	Purpose
Bit to Integer Converter	Map a vector of bits to a corresponding vector of integers
dB Conversion	Convert input of Watts or Volts to decibels
Data Mapper	Map integer symbols from one coding scheme to another
Integer to Bit Converter	Map a vector of integers to a vector of bits

Alphabetical List of Blocks

A-Law Compressor	4-47
A-Law Expander	4-49
Algebraic Deinterleaver	4-51
Algebraic Interleaver	4-53
APP Decoder	4-56
AWGN Channel	4-60
Baseband PLL	4-64
BCH Decoder	4-66
BCH Encoder	4-68
Bernoulli Random Binary Generator	4-70
Binary Cyclic Decoder	4-73
Binary Cyclic Encoder	4-75
Binary-Input RS Encoder	4-77
Binary Linear Decoder	4-79
Binary Linear Encoder	4-81
Binary-Output RS Decoder	4-82
Binary Symmetric Channel	4-84
Binary Vector Noise Generator	4-86
Bit to Integer Converter	4-89
BPSK Demodulator Baseband	4-90
BPSK Modulator Baseband	4-92
Charge Pump PLL	4-94
Complex Phase Difference	4-97
Complex Phase Shift	4-98
Continuous-Time Eye and Scatter Diagrams	4-99
Convolutional Deinterleaver	4-103
Convolutional Encoder	4-105
Convolutional Interleaver	4-107
CPFSK Demodulator Baseband	4-109
CPFSK Demodulator Passband	4-112
CPFSK Modulator Baseband	4-115
CPFSK Modulator Passband	4-118
CPM Demodulator Baseband	4-121
CPM Demodulator Passband	4-125
CPM Modulator Baseband	4-130

CPM Modulator Passband	4-134
Data Mapper	4-139
DBPSK Demodulator Baseband	4-142
DBPSK Modulator Baseband	4-144
Deinterlacer	4-146
Derepeat	4-147
Descrambler	4-150
Differential Decoder	4-152
Differential Encoder	4-153
Discrete Modulo Integrator	4-154
Discrete-Time Eye and Scatter Diagrams	4-156
Discrete-Time VCO	4-159
DPCM Decoder	4-161
DPCM Encoder	4-163
DQPSK Demodulator Baseband	4-165
DQPSK Modulator Baseband	4-167
DSB AM Demodulator Baseband	4-171
DSB AM Demodulator Passband	4-173
DSB AM Modulator Baseband	4-175
DSB AM Modulator Passband	4-176
DSBSC AM Demodulator Baseband	4-178
DSBSC AM Demodulator Passband	4-180
DSBSC AM Modulator Baseband	4-182
DSBSC AM Modulator Passband	4-183
Enabled Quantizer Encode	4-185
Error Rate Calculation	4-187
FM Demodulator Baseband	4-194
FM Demodulator Passband	4-196
FM Modulator Baseband	4-198
FM Modulator Passband	4-200
Gaussian Noise Generator	4-202
General Block Deinterleaver	4-205
General Block Interleaver	4-207
General Multiplexed Deinterleaver	4-208
General Multiplexed Interleaver	4-210
General QAM Demodulator Baseband	4-212
General QAM Demodulator Passband	4-214

General QAM Modulator Baseband	4-217
General QAM Modulator Passband	4-219
GMSK Demodulator Baseband	4-222
GMSK Demodulator Passband	4-225
GMSK Modulator Baseband	4-228
GMSK Modulator Passband	4-231
Hamming Decoder	4-234
Hamming Encoder	4-236
Helical Deinterleaver	4-238
Helical Interleaver	4-241
Insert Zero	4-244
Integer-Input RS Encoder	4-246
Integer-Output RS Decoder	4-248
Integer to Bit Converter	4-250
Integrate and Dump	4-251
Interlacer	4-253
Linearized Baseband PLL	4-254
Matrix Deinterleaver	4-256
Matrix Helical Scan Deinterleaver	4-257
Matrix Helical Scan Interleaver	4-259
Matrix Interleaver	4-262
M-DPSK Demodulator Baseband	4-264
M-DPSK Demodulator Passband	4-267
M-DPSK Modulator Baseband	4-270
M-DPSK Modulator Passband	4-274
M-FSK Demodulator Baseband	4-277
M-FSK Demodulator Passband	4-280
M-FSK Modulator Baseband	4-283
M-FSK Modulator Passband	4-286
Modulo Integrator	4-289
M-PAM Demodulator Baseband	4-290
M-PAM Demodulator Passband	4-293
M-PAM Modulator Baseband	4-297
M-PAM Modulator Passband	4-301
M-PSK Demodulator Baseband	4-305
M-PSK Demodulator Passband	4-308
M-PSK Modulator Baseband	4-311

M-PSK Modulator Passband	4-316
MSK Demodulator Baseband	4-319
MSK Demodulator Passband	4-321
MSK Modulator Baseband	4-324
MSK Modulator Passband	4-326
Mu-Law Compressor	4-329
Mu-Law Expander	4-330
Multipath Rayleigh Fading Channel	4-331
OQPSK Demodulator Baseband	4-334
OQPSK Demodulator Passband	4-336
OQPSK Modulator Baseband	4-339
OQPSK Modulator Passband	4-342
Phase-Locked Loop	4-345
PM Demodulator Baseband	4-348
PM Demodulator Passband	4-350
PM Modulator Baseband	4-352
PM Modulator Passband	4-353
PN Sequence Generator	4-355
Poisson Int Generator	4-358
Puncture	4-360
QPSK Demodulator Baseband	4-362
QPSK Modulator Baseband	4-364
Quantizer Decode	4-367
Random Deinterleaver	4-368
Random-Integer Generator	4-369
Random Interleaver	4-372
Rayleigh Noise Generator	4-373
Rectangular QAM Demodulator Baseband	4-376
Rectangular QAM Demodulator Passband	4-379
Rectangular QAM Modulator Baseband	4-383
Rectangular QAM Modulator Passband	4-387
Rician Fading Channel	4-391
Rician Noise Generator	4-394
Sampled Quantizer Encode	4-397
Scrambler	4-399
SSB AM Demodulator Baseband	4-401
SSB AM Demodulator Passband	4-403

SSB AM Modulator Baseband 4-405

SSB AM Modulator Passband 4-408

Triggered Read From File 4-411

Triggered Write to File 4-414

Uniform Noise Generator 4-416

Viterbi Decoder 4-419

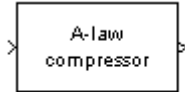
Voltage-Controlled Oscillator 4-424

Windowed Integrator 4-426

Purpose Implement A-law compressor for source coding

Library Source Coding

Description The A-Law Compressor block implements an A-law compressor for the input signal. The formula for the A-law compressor is



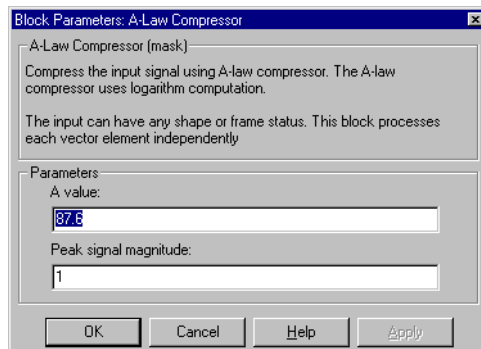
$$y = \begin{cases} \frac{A|x|}{1 + \log A} \operatorname{sgn}(x) & \text{for } 0 \leq |x| \leq \frac{V}{A} \\ \frac{V(1 + \log(A|x|/V))}{1 + \log A} \operatorname{sgn}(x) & \text{for } \frac{V}{A} < |x| \leq V \end{cases}$$

where A is the A-law parameter of the compressor, V is the peak signal magnitude for x , \log is the natural logarithm, and sgn is the signum function (sign in MATLAB).

The most commonly used A value is 87.6.

The input can have any shape or frame status. This block processes each vector element independently.

Dialog Box



A value

The A-law parameter of the compressor.

A-Law Compressor

Peak signal magnitude

The peak value of the input signal. This is also the peak value of the output signal.

Pair Block A-Law Expander

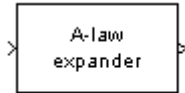
See Also Mu-Law Compressor

References [1] Sklar, Bernard. *Digital Communications: Fundamentals and Applications*. Englewood Cliffs, N.J.: Prentice-Hall, 1988.

Purpose Implement A-law expander for source coding

Library Source Coding

Description

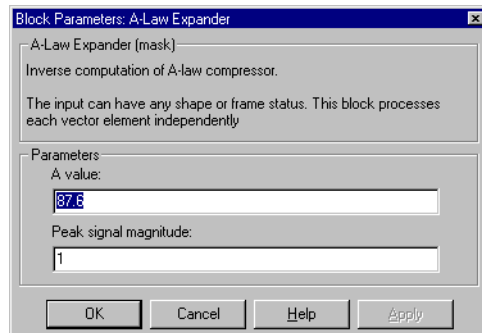


The A-Law Expander block recovers data that the A-Law Compressor block compressed. The formula for the A-law expander, shown below, is the inverse of the compressor function.

$$x = \begin{cases} \frac{y(1 + \log A)}{A} & \text{for } 0 \leq |y| \leq \frac{V}{1 + \log A} \\ e^{|y|(1 + \log A)/V - 1} \frac{V}{A} \text{sgn}(y) & \text{for } \frac{V}{1 + \log A} < |y| \leq V \end{cases}$$

The input can have any shape or frame status. This block processes each vector element independently.

Dialog Box



A value

The A-law parameter of the compressor.

Peak signal magnitude

The peak value of the input signal. This is also the peak value of the output signal.

Match these parameters to the ones in the corresponding A-Law Compressor block.

A-Law Expander

Pair Block A-Law Compressor

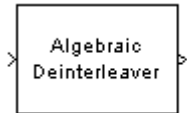
See Also Mu-Law Expander

References [1] Sklar, Bernard. *Digital Communications: Fundamentals and Applications*. Englewood Cliffs, N.J.: Prentice-Hall, 1988.

Purpose Restore ordering of the input symbols using algebraically derived permutation

Library Block sublibrary of Interleaving

Description

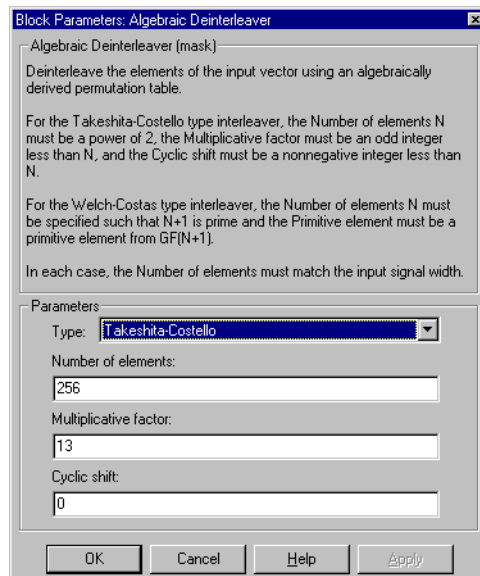


The Algebraic Deinterleaver block restores the original ordering of a sequence that was interleaved using the Algebraic Interleaver block. In typical usage, the parameters in the two blocks have the same values.

The **Number of elements** parameter, N , indicates how many numbers are in the input vector. If the input is frame-based, then it must be a column vector.

The **Type** parameter indicates the algebraic method that the block uses to generate the appropriate permutation table. Choices are **Takeshita-Costello** and **Welch-Costas**. Each of these methods has parameters and restrictions that are specific to it; these are described on the reference page for the Algebraic Interleaver block.

Dialog Box



Type

The type of permutation table that the block uses for deinterleaving. Choices are **Takeshita-Costello** and **Welch-Costas**.

Algebraic Deinterleaver

Number of elements

The number of elements, N, in the input vector.

Multiplicative factor

The factor used to compute the corresponding interleaver's cycle vector.

This field appears only if **Type** is set to **Takeshita-Costello**.

Cyclic shift

The amount by which the block shifts indices when creating the corresponding interleaver's permutation table. This field appears only if **Type** is set to **Takeshita-Costello**.

Primitive element

An element of order N in the finite field $GF(N+1)$. This field appears only if **Type** is set to **Welch-Costas**.

Pair Block

Algebraic Interleaver

See Also

General Block Deinterleaver

References

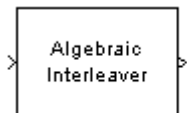
[1] Heegard, Chris and Stephen B. Wicker. *Turbo Coding*. Boston: Kluwer Academic Publishers, 1999.

[2] Takeshita, O. Y. and D. J. Costello, Jr. "New Classes Of Algebraic Interleavers for Turbo-Codes." *Proc. 1998 IEEE International Symposium on Information Theory*, Boston, Aug. 16-21, 1998. 419.

Purpose Reorder the input symbols using algebraically derived permutation table

Library Block sublibrary of Interleaving

Description



The Algebraic Interleaver block rearranges the elements of its input vector using a permutation that is algebraically derived. The **Number of elements** parameter, N , indicates how many numbers are in the input vector. If the input is frame-based, then it must be a column vector.

The **Type** parameter indicates the algebraic method that the block uses to generate the appropriate permutation table. Choices are **Takeshita-Costello** and **Welch-Costas**. Each of these methods has parameters and restrictions that are specific to it:

- If **Type** is set to **Welch-Costas**, then $N+1$ must be prime. The **Primitive element** parameter is an integer, A , between 1 and N that represents a primitive element of the finite field $GF(N+1)$. This means that every nonzero element of $GF(N+1)$ can be expressed as A raised to some integer power.

In a Welch-Costas interleaver, the permutation maps the integer k to $\text{mod}(A^k, N+1) - 1$.

- If **Type** is set to **Takeshita-Costello**, then N must be 2^m for some integer m . The **Multiplicative factor** parameter, h , must be an odd integer less than N . The **Cyclic shift** parameter, k , must be a nonnegative integer less than N .

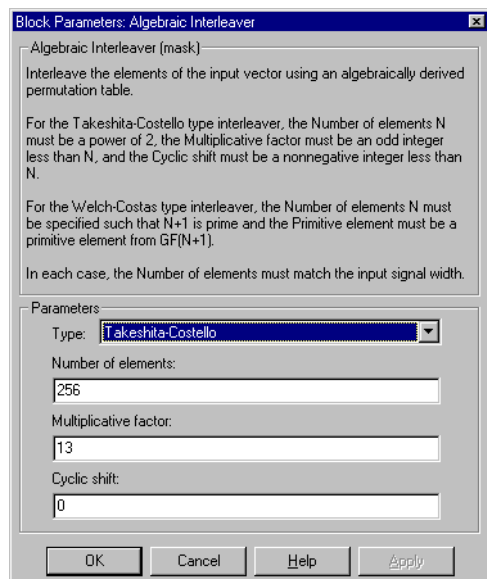
A Takeshita-Costello interleaver uses a length- N *cycle vector* whose n th element is

$$\text{mod}(k*(n-1)*n/2, N)$$

for integers n between 1 and N . The block creates a permutation vector by listing, for each element of the cycle vector in ascending order, one plus the element's successor. The interleaver's actual permutation table is the result of shifting the elements of the permutation vector left by the **Cyclic shift** parameter. (The block performs all computations on numbers and indices modulo N .)

Algebraic Interleaver

Dialog Box



Type

The type of permutation table that the block uses for interleaving.

Number of elements

The number of elements, N , in the input vector.

Multiplicative factor

The factor used to compute the interleaver's cycle vector. This field appears only if **Type** is set to **Takeshita-Costello**.

Cyclic shift

The amount by which the block shifts indices when creating the permutation table. This field appears only if **Type** is set to **Takeshita-Costello**.

Primitive element

An element of order N in the finite field $GF(N+1)$. This field appears only if **Type** is set to **Welch-Costas**.

Pair Block

Algebraic Deinterleaver

See Also General Block Interleaver

- References**
- [1] Heegard, Chris and Stephen B. Wicker. *Turbo Coding*. Boston: Kluwer Academic Publishers, 1999.
 - [2] Takeshita, O. Y. and D. J. Costello, Jr. "New Classes Of Algebraic Interleavers for Turbo-Codes." *Proc. 1998 IEEE International Symposium on Information Theory*, Boston, Aug. 16-21, 1998. 419.

APP Decoder

Purpose Decode a convolutional code using the a posteriori probability (APP) method

Library Convolutional sublibrary of Channel Coding

Description The APP Decoder block performs a posteriori probability (APP) decoding of a convolutional code. You can use this block to build a turbo decoder.



Inputs and Outputs

The input $L(u)$ represents the sequence of log-likelihoods of encoder input bits, while the input $L(c)$ represents the sequence of log-likelihoods of code bits. The outputs $L(u)$ and $L(c)$ are updated versions of these sequences, based on information about the encoder.

If the convolutional code uses an alphabet of 2^n possible symbols, then this block's $L(c)$ vectors have length $Q \cdot n$ for some positive integer Q . Similarly, if the decoded data uses an alphabet of 2^k possible output symbols, then this block's $L(u)$ vectors have length $Q \cdot k$. The integer Q is the number of frames that the block processes in each step.

The inputs can be either:

- Sample-based vectors having the same dimension and orientation, with $Q = 1$
- Frame-based column vectors with any positive integer for Q

If you only need the input $L(c)$ and output $L(u)$, then you can attach a Simulink Ground block to the input $L(u)$ and a Simulink Terminator block to the output $L(c)$.

Specifying the Encoder

To define the convolutional encoder that produced the coded input, use the **Trellis structure** parameter. This parameter is a MATLAB structure whose format is described in the section, “Trellis Description of a Convolutional Encoder,” in the *Communications Toolbox User's Guide*. You can use this parameter field in two ways:

- If you have a variable in the MATLAB workspace that contains the trellis structure, then enter its name as the **Trellis structure** parameter. This way is preferable because it causes Simulink to spend less time updating the

diagram at the beginning of each simulation, compared to the usage in the next bulleted item.

- If you want to specify the encoder using its constraint length, generator polynomials, and possibly feedback connection polynomials, then use a `poly2trellis` command within the **Trellis structure** field. For example, to use an encoder with a constraint length of 7, code generator polynomials of 171 and 133 (in octal numbers), and a feedback connection of 171 (in octal), set the **Trellis structure** parameter to

```
poly2trellis(7, [171 133], 171)
```

To indicate how the encoder treats the trellis at the beginning and end of each frame, set the **Termination method** parameter to either **Truncated** or **Terminated**. The **Truncated** option indicates that the encoder resets to the all-zeros state at the beginning of each frame, while the **Terminated** option indicates that the encoder forces the trellis to end each frame in the all-zeros state. If you use the Convolutional Encoder block with the **Reset** parameter set to **On each frame**, then use the **Truncated** option in this block.

Specifying Details of the Algorithm

You can control part of the decoding algorithm using the **Algorithm** parameter. The **True APP** option implements a posteriori probability. To gain speed, both the **Max*** and **Max** options approximate expressions like

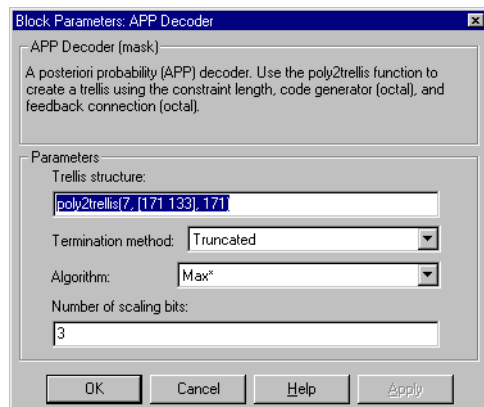
$$\log \sum_i \exp a_i$$

by other quantities. The **Max** option uses $\max\{a_i\}$ as the approximation, while the **Max*** option uses $\max\{a_i\}$ plus a correction term.

The **Max*** option enables the **Scaling bits** parameter in the mask. This parameter is the number of bits by which the block scales the data it processes internally. You can use this parameter to avoid losing precision during the computations. It is especially appropriate if your implementation uses fixed-point components. For more information about the **Max*** option, see the article by Viterbi in the “References” section below.

APP Decoder

Dialog Box



Trellis structure

MATLAB structure that contains the trellis description of the convolutional encoder.

Termination method

Either **Truncated** or **Terminated**. This parameter indicates how the convolutional encoder treats the trellis at the beginning and end of frames.

Algorithm

Either **True APP**, **Max***, or **Max**.

Number of scaling bits

An integer between 0 and 8 that indicates by how many bits the decoder scales data in order to avoid losing precision. This field is active only when **Algorithm** is set to **Max***.

See Also

Viterbi Decoder, Convolutional Encoder; `poly2trellis` (Communications Toolbox)

References

[1] Benedetto, Sergio and Guido Montorsi. "Performance of Continuous and Blockwise Decoded Turbo Codes." *IEEE Communications Letters*, vol. 1, May 1997. 77-79.

[2] Benedetto, S., G. Montorsi, D. Divsalar, and F. Pollara. "A Soft-Input Soft-Output Maximum A Posteriori (MAP) Module to Decode Parallel and

Serial Concatenated Codes.” *JPL TMO Progress Report*, vol. 42-127, November 1996. [This electronic journal is available at http://tmo.jpl.nasa.gov/tmo/progress_report/index.html.]

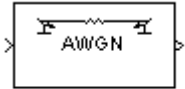
[3] Viterbi, Andrew J. “An Intuitive Justification and a Simplified Implementation of the MAP Decoder for Convolutional Codes.” *IEEE Journal on Selected Areas in Communications*, vol. 16, February 1998. 260-264.

AWGN Channel

Purpose Add white Gaussian noise to the input signal

Library Channels

Description



The AWGN Channel block adds white Gaussian noise to a real or complex input signal. When the input signal is real, this block adds real Gaussian noise and produces a real output signal. When the input signal is complex, this block adds complex Gaussian noise and produces a complex output signal. This block inherits its sample time from the input signal.

This block uses the DSP Blockset's Random Source block to generate the noise. The **Initial seed** parameter in this block initializes the noise generator. **Initial seed** can be either a scalar or a vector whose length matches the number of channels in the input signal. For details on **Initial seed**, see the Random Source block reference page in the *DSP Blockset User's Guide*.

Frame-Based Processing and Input Dimensions

This block can process multichannel signals that are frame-based or sample-based. The guidelines below indicate how the block interprets your data, depending on the data's shape and frame status:

- If your input is a sample-based scalar, then the block adds scalar Gaussian noise to your signal.
- If your input is a sample-based vector or a frame-based row vector, then the block adds independent Gaussian noise to each channel.
- If your input is a frame-based column vector, then the block adds a frame of Gaussian noise to your single-channel signal.
- If your input is a frame-based m-by-n matrix, then the block adds a length-m frame of Gaussian noise independently to each of the n channels.

The input cannot be a sample-based m-by-n matrix if both m and n are greater than 1.

Specifying the Variance Directly or Indirectly

You can specify the variance of the noise generated by the AWGN Channel block using one of four modes:

- **Signal to noise ratio (Es/No)**, where the block calculates the variance from these quantities that you specify in the block mask:

- **Es/No**, the ratio of signal energy to noise power spectral density
- **Input signal power**, the power of the input symbols
- **Symbol period**
- **Signal to noise ratio (SNR)**, where the block calculates the variance from these quantities that you specify in the block mask:
 - **SNR**, the ratio of signal power to noise power
 - **Input signal power**, the power of the input samples
- **Variance from mask**, where you specify the variance in the block mask. The value must be positive.
- **Variance from port**, where you provide the variance as an input to the block. The variance input must be positive, and its sampling rate must equal that of the input signal. If the first input signal is sample-based, then the variance input must be sample-based. If the first input signal is frame-based, then the variance input can be either frame-based with exactly one row, or sample-based.

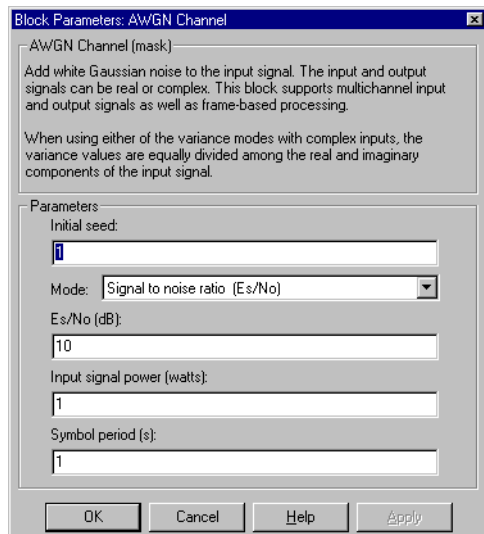
In both **Variance from mask** mode and **Variance from port** mode, these rules describe how the block interprets the variance:

- If the variance is a scalar, then all signal channels are uncorrelated but share the same variance.
- If the variance is a vector whose length is the number of channels in the input signal, then each element represents the variance of the corresponding signal channel.

Note If you apply complex input signals to the AWGN Channel block, then it adds complex zero-mean Gaussian noise with the calculated or specified variance. The variance of each of the quadrature components of the complex noise is half of the calculated or specified value.

AWGN Channel

Dialog Box



Initial seed

The seed for the Gaussian noise generator.

Mode

The mode by which you specify the noise variance: **Signal to noise ratio (Es/No)**, **Signal to noise ratio (SNR)**, **Variance from mask**, or **Variance from port**.

Es/No (dB)

The ratio of signal energy per symbol to noise power spectral density, in decibels. This field appears only if **Mode** is set to **Es/No**.

SNR (dB)

The ratio of signal power to noise power, in decibels. This field appears only if **Mode** is set to **SNR**.

Input signal power (watts)

The root mean square power of the input symbols (if **Mode** is **Es/No**) or input samples (if **Mode** is **SNR**), in watts. This field appears only if **Mode** is set to either **Es/No** or **SNR**.

Symbol period (s)

The duration of a channel symbol, in seconds. This field appears only if **Mode** is set to **Es/No**.

Variance

The variance of the white Gaussian noise. This field appears only if **Mode** is set to **Variance from mask**.

See Also

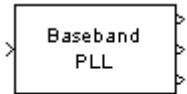
Random Source (DSP Blockset)

Baseband PLL

Purpose Implement a baseband phase-locked loop

Library Synchronization

Description



The Baseband PLL (phase-locked loop) block is a feedback control system that automatically adjusts the phase of a locally generated signal to match the phase of an input signal. Unlike the Phase-Locked Loop block, this block uses a baseband method and does not depend on a carrier frequency.

This PLL has these three components:

- An integrator used as a phase detector.
- A filter. You specify the filter's transfer function using the **Lowpass filter numerator** and **Lowpass filter denominator** mask parameters. Each is a vector that gives the respective polynomial's coefficients in order of descending powers of s .

To design a filter, you can use functions such as `butler`, `cheby1`, and `cheby2` in the Signal Processing Toolbox. The default filter is a Chebyshev type II filter whose transfer function arises from the command below.

```
[num, den] = cheby2(3, 40, 100, 's')
```

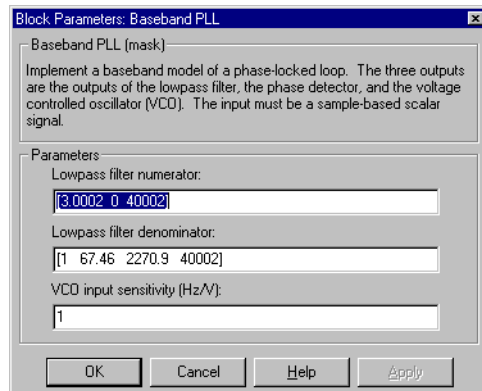
- A voltage-controlled oscillator (VCO). You specify the sensitivity of the VCO signal to its input using the **VCO input sensitivity** parameter. This parameter, measured in Hertz per volt, is a scale factor that determines how much the VCO shifts from its quiescent frequency.

The input signal represents the received signal. The input must be a sample-based scalar signal. The three output ports produce:

- The output of the filter
- The output of the phase detector
- The output of the VCO

This model is nonlinear; for a linearized version, use the Linearized Baseband PLL block.

Dialog Box



Lowpass filter numerator

The numerator of the lowpass filter's transfer function, represented as a vector that lists the coefficients in order of descending powers of s .

Lowpass filter denominator

The denominator of the lowpass filter's transfer function, represented as a vector that lists the coefficients in order of descending powers of s .

VCO input sensitivity (Hz/V)

This value scales the input to the VCO and, consequently, the shift from the VCO's quiescent frequency.

See Also

Linearized Baseband PLL, Phase-Locked Loop

References

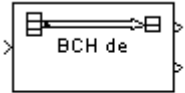
For more information about phase-locked loops, see the works listed in "Selected Bibliography for Synchronization" on page 2-92.

BCH Decoder

Purpose Decode a BCH code to recover binary vector data

Library Block sublibrary of Channel Coding

Description



The BCH Decoder block recovers a binary message vector from a binary BCH codeword vector. For proper decoding, the first two parameter values in this block should match the parameters in the corresponding BCH Encoder block.

The input is the binary codeword vector and the first output is the corresponding binary message vector. If the BCH code has message length K and codeword length N , then the input has length N and the first output has length K . If the input is frame-based, then it must be a column vector.

The number N must have the form $2^M - 1$, where M is an integer greater than or equal to 3. For a given codeword length N , only specific message lengths K are valid for a BCH code. To see which values of K are valid, use the `bchpoly` function in the Communications Toolbox. No known analytic formula describes the relationship among the codeword length, message length, and error-correction capability.

The second output is the number of errors detected during decoding of the codeword. A negative integer indicates that the block detected more errors than it could correct using the coding scheme.

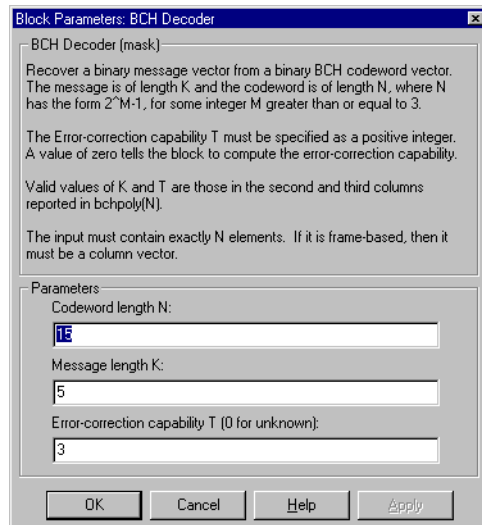
The sample times of all input and output signals are equal.

The **Error-correction capability T** parameter either:

- Indicates the error-correction capability of the code as a positive integer, or
- Tells the block to compute the error-correction capability, if you enter zero

The block runs faster in the first case above. You can use the `bchpoly` function in the Communications Toolbox to calculate the error-correction capability.

Dialog Box



Codeword length N

The codeword length, which is also the vector length of the first input.

Message length K

The message length, which is also the vector length of the first output.

Error-correction capability T

Either the error-correction capability of the code, or zero. A zero forces the block to calculate the error-control capability when initializing.

Pair Block

BCH Encoder

See Also

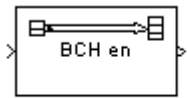
bchpoly (Communications Toolbox)

BCH Encoder

Purpose Create a BCH code from binary vector data

Library Block sublibrary of Channel Coding

Description



The BCH Encoder block creates a BCH code with message length K and codeword length N . You specify both N and K directly in the block mask.

The input must contain exactly K elements. If it is frame-based, then it must be a column vector. The output is a vector of length N .

N must have the form 2^M-1 , where M is an integer greater than or equal to 3. For a given codeword length N , only specific message lengths K are valid for a BCH code. To see which values of K are valid, use the `bchpoly` function in the Communications Toolbox. For example, in the output below, the second column lists all possible message lengths that correspond to a codeword length of 15. The third column lists the corresponding error-correction capabilities.

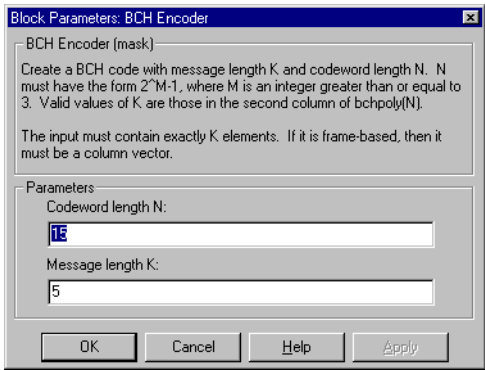
```
params = bchpoly(15)
```

```
params =
```

15	11	1
15	7	2
15	5	3

No known analytic formula describes the relationship among the codeword length, message length, and error-correction capability.

Dialog Box



Codeword length N

The codeword length, which is also the output vector length.

Message length K

The message length, which is also the input vector length.

Pair Block

BCH Decoder

See Also

`bchpoly` (Communications Toolbox)

Bernoulli Random Binary Generator

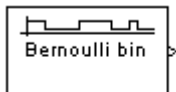
Purpose

Generate Bernoulli-distributed random binary numbers

Library

Comm Sources

Description



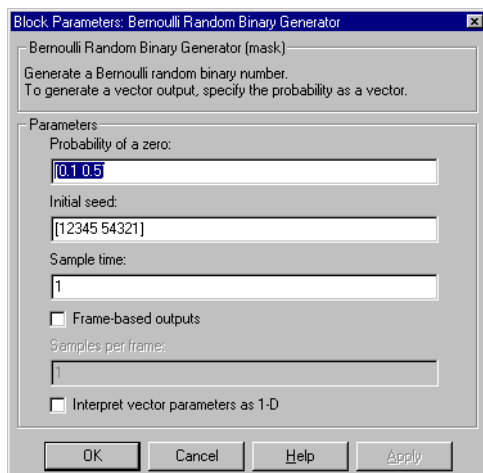
The Bernoulli Random Binary Generator block generates random binary numbers using a Bernoulli distribution. The Bernoulli distribution with parameter p produces zero with probability p and one with probability $1-p$. The Bernoulli distribution has mean value $1-p$ and variance $p(1-p)$. The **Probability of a zero** parameter specifies p , and can be any real number between zero and one.

Attributes of Output Signal

The output signal can be a frame-based matrix, a sample-based row or column vector, or a sample-based one-dimensional array. These attributes are controlled by the **Frame-based outputs**, **Samples per frame**, and **Interpret vector parameters as 1-D** parameters. See “Signal Attribute Parameters for Random Sources” on page 2-7 for more details.

The number of elements in the **Initial seed** and **Probability of a zero** parameters becomes the number of columns in a frame-based output or the number of elements in a sample-based vector output. Also, the shape (row or column) of the **Initial seed** and **Probability of a zero** parameters becomes the shape of a sample-based two-dimensional output signal.

Dialog Box



Probability of a zero

The probability with which a zero output occurs.

Initial seed

The initial seed value for the random number generator. The seed can be either a vector of the same length as the **Probability of a zero** parameter, or a scalar.

Sample time

The period of each sample-based vector or each row of a frame-based matrix.

Frame-based outputs

Determines whether the output is frame-based or sample-based. This box is active only if **Interpret vector parameters as 1-D** is unchecked.

Samples per frame

The number of samples in each column of a frame-based output signal. This field is active only if **Frame-based outputs** is checked.

Bernoulli Random Binary Generator

Interpret vector parameters as 1-D

If this box is checked, then the output is a one-dimensional signal.

Otherwise, the output is a two-dimensional signal. This box is active only if **Frame-based outputs** is unchecked.

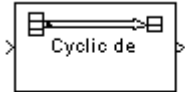
See Also

Binary Vector Noise Generator, Random-Integer Generator, Binary Symmetric Channel; `randint` (Communications Toolbox), `rand` (built-in MATLAB function)

Purpose Decode a systematic cyclic code to recover binary vector data

Library Block sublibrary of Channel Coding

Description



The Binary Cyclic Decoder block recovers a message vector from a codeword vector of a binary systematic cyclic code. For proper decoding, the parameter values in this block should match those in the corresponding Binary Cyclic Encoder block.

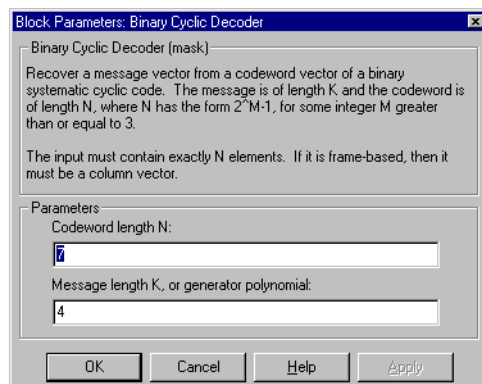
If the cyclic code has message length K and codeword length N , then N must have the form $2^M - 1$ for some integer M greater than or equal to 3.

The input must contain exactly N elements. If it is frame-based, then it must be a column vector. The output is a vector of length K .

You can determine the systematic cyclic coding scheme in one of two ways:

- To create an $[N, K]$ code, enter N and K as the first and second mask parameters, respectively. The block computes an appropriate generator polynomial, namely, $\text{cycl pol y}(N, K, ' \text{min}')$.
- To create a code with codeword length N and a particular degree- $(N-K)$ binary *generator polynomial*, enter N as the first parameter and a binary vector as the second parameter. The vector represents the generator polynomial by listing its coefficients in order of ascending exponents. You can create cyclic generator polynomials using the cycl pol y function in the Communications Toolbox.

Dialog Box



Binary Cyclic Decoder

Codeword length N

The codeword length N, which is also the input vector length.

Message length K, or generator polynomial

Either the message length, which is also the output vector length; or a binary vector that represents the generator polynomial for the code.

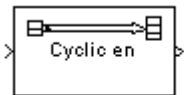
Pair Block Binary Cyclic Encoder

See Also `cyclpoly` (Communications Toolbox)

Purpose Create a systematic cyclic code from binary vector data

Library Block sublibrary of Channel Coding

Description The Binary Cyclic Encoder block creates a systematic cyclic code with message length K and codeword length N . The number N must have the form $2^M - 1$, where M is an integer greater than or equal to 3.

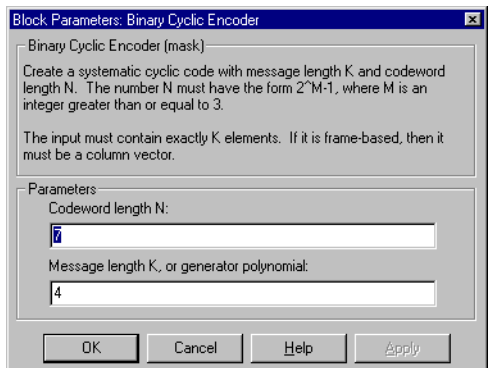


The input must contain exactly K elements. If it is frame-based, then it must be a column vector. The output is a vector of length N .

You can determine the systematic cyclic coding scheme in one of two ways:

- To create an $[N, K]$ code, enter N and K as the first and second mask parameters, respectively. The block computes an appropriate generator polynomial, namely, `cycl poly(N, K, 'mi n')`.
- To create a code with codeword length N and a particular degree- $(N-K)$ binary *generator polynomial*, enter N as the first parameter and a binary vector as the second parameter. The vector represents the generator polynomial by listing its coefficients in order of ascending exponents. You can create cyclic generator polynomials using the `cycl poly` function in the Communications Toolbox.

Dialog Box



Codeword length N

The codeword length, which is also the output vector length.

Binary Cyclic Encoder

Message length K, or generator polynomial

Either the message length, which is also the input vector length; or a binary vector that represents the generator polynomial for the code.

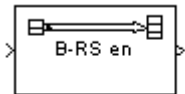
Pair Block Binary Cyclic Decoder

See Also cycl pol y (Communications Toolbox)

Purpose Create a Reed-Solomon code from binary vector data

Library Block sublibrary of Channel Coding

Description



The Binary-Input RS Encoder block creates a Reed-Solomon code with message length K and codeword length N . You specify both N and K directly in the block mask. N must have the form $2^M - 1$, where M is an integer greater than or equal to 3. The code is more efficient if $N - K$ is an even integer.

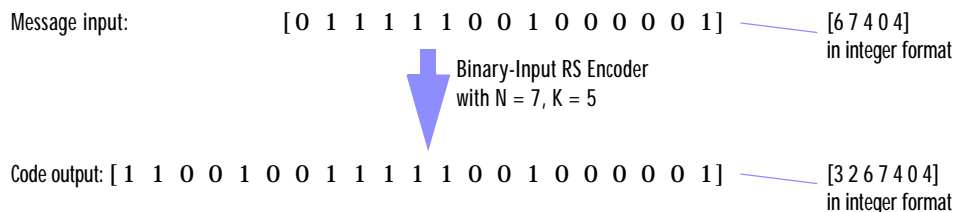
The input and output are binary-valued signals that represent messages and codewords, respectively. The input must contain exactly $M \cdot K$ elements. If it is frame-based, then it must be a column vector. The output is a vector of length $M \cdot N$.

The $M \cdot K$ input bits represent K integers between 0 and $2^M - 1$, where more significant bits are to the right. Similarly, the $M \cdot N$ output bits represent N integers between 0 and $2^M - 1$. These integers in turn represent elements of the finite field $GF(2^M)$.

An (N, K) Reed-Solomon code can correct up to $\text{floor}((N - K) / 2)$ symbol errors (*not* bit errors) in each codeword.

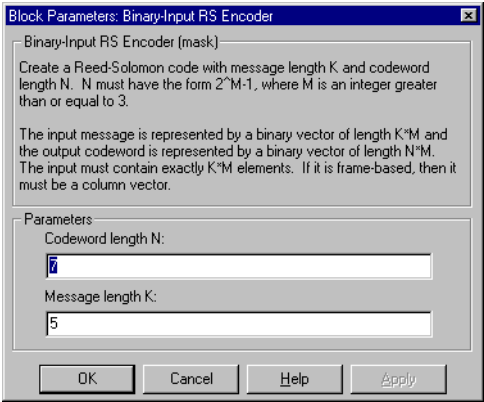
Examples

Suppose $M = 3$, $N = 2^3 - 1 = 7$, and $K = 5$. Then a message is a binary vector of length 15 that represents 5 three-bit integers. A corresponding codeword is a binary vector of length 21 that represents 7 three-bit integers. The figure below shows the codeword that would result from a particular message word. The integer format equivalents illustrate that the highest-order bit is at the right.



Binary-Input RS Encoder

Dialog Box



Codeword length N

The codeword length. The output has vector length $M*N$.

Message length K

The message length. The input has vector length $M*K$.

Pair Block

Binary-Output RS Decoder

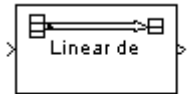
See Also

Integer-Input RS Encoder

Purpose Decode a linear block code to recover binary vector data

Library Block sublibrary of Channel Coding

Description



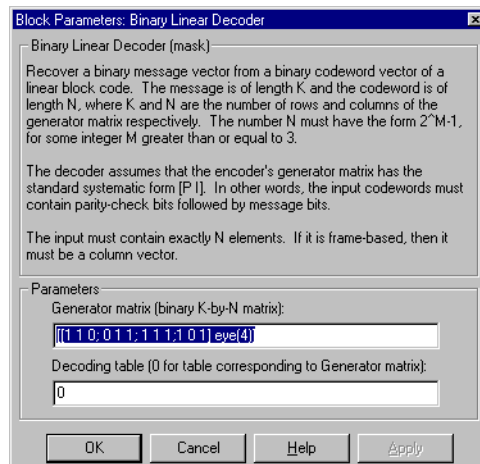
The Binary Linear Decoder block recovers a binary message vector from a binary codeword vector of a linear block code.

The **Generator matrix** parameter is the generator matrix for the block code. For proper decoding, this should match the **Generator matrix** parameter in the corresponding Binary Linear Encoder block. If N is the codeword length of the code, then **Generator matrix** must have N columns. If K is the message length of the code, then the **Generator matrix** parameter must have K rows.

The input must contain exactly N elements. If it is frame-based, then it must be a column vector. The output is a vector of length K .

The decoder tries to correct errors, using the **Decoding table** parameter. If **Decoding table** is the scalar 0, then the block defaults to the table produced by the Communications Toolbox function `syndtbl`. Otherwise, **Decoding table** must be a 2^{N-K} -by- N binary matrix. The r th row of this matrix is the correction vector for a received binary codeword whose syndrome has decimal integer value $r-1$. The syndrome of a received codeword is its product with the transpose of the parity-check matrix.

Dialog Box



Binary Linear Decoder

Generator matrix

Generator matrix for the code; same as in Binary Linear Encoder block.

Decoding table

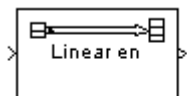
Either a 2^{N-K} -by- N matrix that lists correction vectors for each codeword's syndrome; or the scalar 0, in which case the block defaults to the table corresponding to the **Generator matrix** parameter.

Pair Block Binary Linear Encoder

Purpose Create a linear block code from binary vector data

Library Block sublibrary of Channel Coding

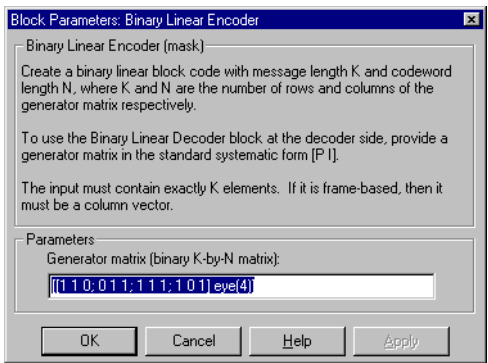
Description



The Binary Linear Encoder block creates a binary linear block code using a generator matrix that you provide in the parameter mask. If K is the message length of the code, then the **Generator matrix** parameter must have K rows. If N is the codeword length of the code, then **Generator matrix** must have N columns.

The input must contain exactly K elements. If it is frame-based, then it must be a column vector. The output is a vector of length N .

Dialog Box



Generator matrix

A K -by- N matrix, where K is the message length and N is the codeword length.

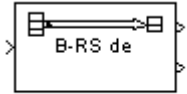
Pair Block Binary Linear Decoder

Binary-Output RS Decoder

Purpose Decode a Reed-Solomon code to recover binary vector data

Library Block sublibrary of Channel Coding

Description



The Binary-Output RS Decoder block recovers a binary message vector from a binary Reed-Solomon codeword vector. For proper decoding, the parameter values in this block should match those in the corresponding Binary-Input RS Encoder block.

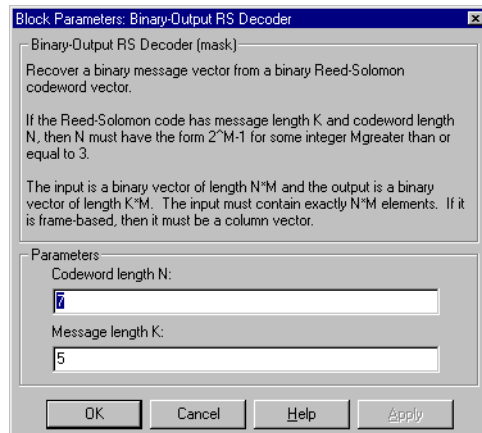
If the Reed-Solomon code has message length K and codeword length N , then N must have the form $2^M - 1$ for some integer M greater than or equal to 3. The code is more efficient if $N - K$ is an even integer.

The input and first output are binary-valued signals that represent codewords and messages, respectively. The input must contain exactly $M \cdot N$ elements. If it is frame-based, then it must be a column vector. The first output is a vector of length $M \cdot K$.

The $M \cdot N$ input bits represent N integers between 0 and $2^M - 1$, where more significant bits are to the right. Similarly, $M \cdot K$ output bits represent K integers between 0 and $2^M - 1$. These integers in turn represent elements of the finite field $GF(2^M)$.

The second output is the number of errors detected during decoding of the codeword. A negative integer indicates that the block detected more errors than it could correct using the coding scheme. An (N, K) Reed-Solomon code can correct up to $\text{floor}((N - K) / 2)$ symbol errors (*not* bit errors) in each codeword.

Dialog Box



Codeword length N

The codeword length. The input has vector length $M \times N$.

Message length K

The message length. The first output has vector length $M \times K$.

Pair Block

Binary-Input RS Encoder

See Also

Integer-Output RS Decoder

Binary Symmetric Channel

Purpose Introduce binary errors

Library Channels

Description

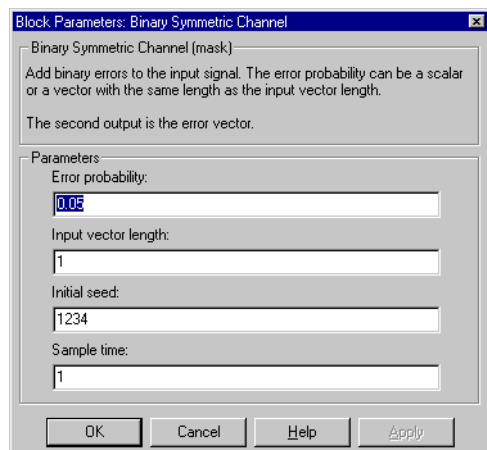


The Binary Symmetric Channel block introduces binary errors to the signal transmitted through this channel.

The input port is the transmitted binary signal. The input can be either a scalar, a sample-based vector, or a frame-based row vector. This block processes each vector element independently, and introduces an error in a given spot with probability **Error probability**.

The first output port is the binary signal that has passed through the channel. The second output port is the vector of errors that were introduced.

Dialog Box



Error probability

The probability that a binary error will occur. The value of this parameter must be between zero and one.

Input vector length

Length of the input vector signal. This is the same as the vector length of the signal at the first output port.

Initial seed

The initial seed value for the random number generator.

Sample time

The block's sample time.

See Also

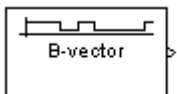
Bernoulli Random Binary Generator

Binary Vector Noise Generator

Purpose Generate a binary vector while controlling the number of 1s

Library Comm Sources

Description



The Binary Vector Noise Generator outputs a random binary vector whose length is the **Binary vector length** parameter. The **Probabilities** parameter helps determine how many 1s appear in each output vector. Once the number of 1s is determined, their placement is determined according to a uniform distribution.

If p_1, p_2, \dots, p_m are the entries in the **Probabilities** parameter, then p_1 is the probability that the output vector will have a single 1, p_2 is the probability that the output vector will have exactly two 1s, and so on. Note that **Probabilities** must have sum less than or equal to one, and length less than or equal to the **Binary vector length**. Also, the probability of a zero vector is one minus the sum of **Probabilities**.

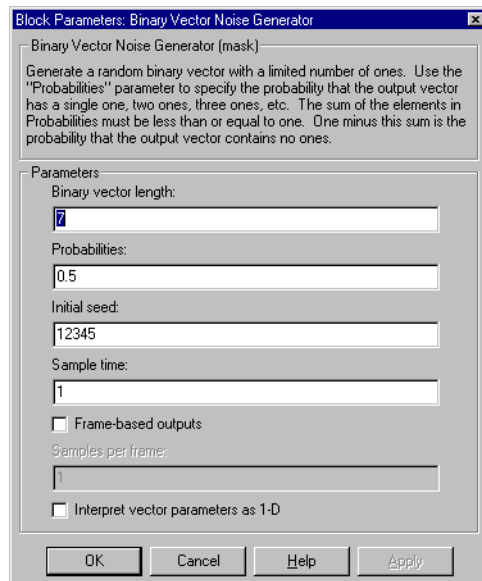
This block is useful in testing error-control coding algorithms.

Attributes of Output Signal

The output signal can be a frame-based matrix, a sample-based row or column vector, or a sample-based one-dimensional array. These attributes are controlled by the **Frame-based outputs**, **Samples per frame**, and **Interpret vector parameters as 1-D** parameters. See “Signal Attribute Parameters for Random Sources” on page 2-7 for more details.

The **Binary vector length** parameter becomes the number of columns in a frame-based output or the number of elements in a sample-based vector output. Also, the shape (row or column) of the **Probabilities** parameter becomes the shape of a sample-based two-dimensional output signal.

Dialog Box



Binary vector length

The output vector length.

Probabilities

A vector whose k th entry indicates the probability that the output vector has exactly k 1s.

Initial seed

The initial seed value for the random number generator. This must be a

Sample time

The period of each sample-based vector or each row of a frame-based matrix.

Frame-based outputs

Determines whether the output is frame-based or sample-based. This box is active only if **Interpret vector parameters as 1-D** is unchecked.

Binary Vector Noise Generator

Samples per frame

The number of samples in each column of a frame-based output signal. This field is active only if **Frame-based outputs** is checked.

Interpret vector parameters as 1-D

If this box is checked, then the output is a one-dimensional signal. Otherwise, the output is a two-dimensional signal. This box is active only if **Frame-based outputs** is unchecked.

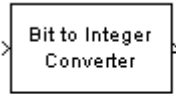
See Also

Bernoulli Random Binary Generator; randerr (Communications Toolbox)

Purpose Map a vector of bits to a corresponding vector of integers

Library Utility Functions

Description

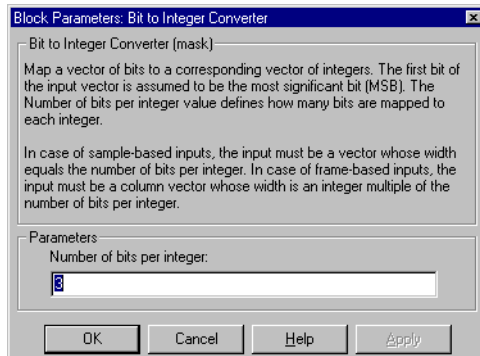


The Bit to Integer Converter block maps groups of bits in the input vector to integers in the output vector. If **M** is the **Number of bits per integer** parameter, then the block maps each group of **M** bits to an integer between 0 and $2^M - 1$. As a result, the output vector length is $1/M$ times the input vector length.

If the input is sample-based input, then it must be a vector whose length equals the **Number of bits per integer** parameter. If the input is frame-based, then it must be a column vector whose length is an integer multiple of **Number of bits per integer**.

The block interprets the first bit in each group as the most significant bit.

Dialog Box



Number of bits per integer

The number of input bits that the block maps to each integer of the output. This parameter must be an integer between 1 and 31.

Examples

If the input is [0; 1; 1; 1; 1; 1; 0; 1] and the **Number of bits per integer** parameter is 4, then the output is [7; 13]. The block maps the first group of four bits (0, 1, 1, 1) to 7 and the second group of four bits (1, 1, 0, 1) to 13. Notice that the output length is one-fourth of the output length.

Pair Block

Integer to Bit Converter

BPSK Demodulator Baseband

Purpose

Demodulate BPSK-modulated data

Library

PM, in Digital Baseband sublibrary of Modulation

Description



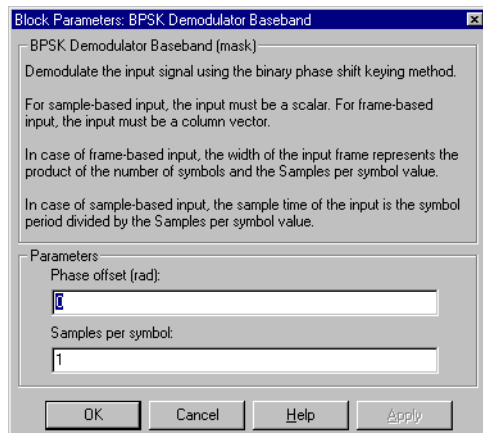
The BPSK Demodulator Baseband block demodulates a signal that was modulated using the binary phase shift keying method. The input is a baseband representation of the modulated signal. The input can be either a scalar or a frame-based column vector.

The input must be a discrete-time complex signal. The block maps the points $\exp(j\theta)$ and $-\exp(j\theta)$ to 0 and 1, respectively, where θ is the **Phase offset** parameter.

Processing an Upsampled Modulated Signal

The input signal can be an upsampled version of the modulated signal. The **Samples per symbol** parameter is the upsampling factor. It must be a positive integer. For more information, see “Upsampled Signals and Rate Changes” on page 2-72.

Dialog Box



Phase offset (rad)

The phase of the zeroth point of the signal constellation.

Samples per symbol

The number of input samples that represent each modulated symbol.

Pair Block BPSK Modulator Baseband

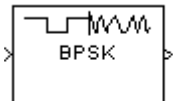
See Also M-PSK Demodulator Baseband, QPSK Demodulator Baseband, DBPSK Demodulator Baseband

BPSK Modulator Baseband

Purpose Modulate using the binary phase shift keying method

Library PM, in Digital Baseband sublibrary of Modulation

Description The BPSK Modulator Baseband block modulates using the binary phase shift keying method. The output is a baseband representation of the modulated signal.

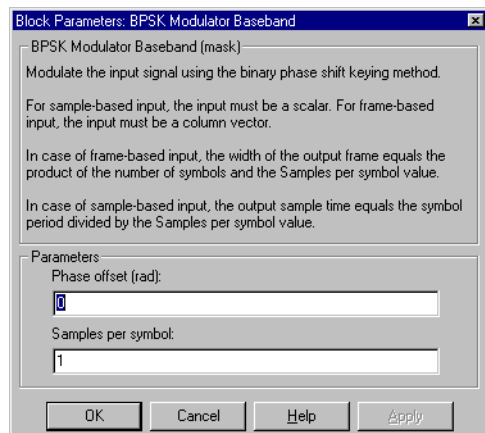


The input must be a discrete-time binary-valued signal. If the input bit is 0 or 1, respectively, then the modulated symbol is $\exp(j\theta)$ or $-\exp(j\theta)$ respectively, where θ is the **Phase offset** parameter.

Upsampling the Modulated Signal

This block can output an upsampled version of the modulated signal. The **Samples per symbol** parameter is the upsampling factor. It must be a positive integer. For more information, see “Upsampled Signals and Rate Changes” on page 2-72.

Dialog Box



Phase offset (rad)

The phase of the zeroth point of the signal constellation.

Samples per symbol

The number of output samples that the block produces for each input bit.

Pair Block BPSK Demodulator Baseband

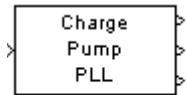
See Also M-PSK Modulator Baseband, QPSK Modulator Baseband, DBPSK Modulator Baseband

Charge Pump PLL

Purpose Implement a charge pump phase-locked loop using a digital phase detector

Library Synchronization

Description



The Charge Pump PLL (phase-locked loop) block automatically adjusts the phase of a locally generated signal to match the phase of an input signal. It is suitable for use with digital signals.

This PLL has these three components:

- A sequential logic phase detector, also called a digital phase detector or a phase/frequency detector.
- A filter. You specify the filter's transfer function using the **Lowpass filter numerator** and **Lowpass filter denominator** mask parameters. Each is a vector that gives the respective polynomial's coefficients in order of descending powers of s .

To design a filter, you can use functions such as `butter`, `cheby1`, and `cheby2` in the Signal Processing Toolbox. The default filter is a Chebyshev type II filter whose transfer function arises from the command below.

```
[num, den] = cheby2(3, 40, 100, 's')
```

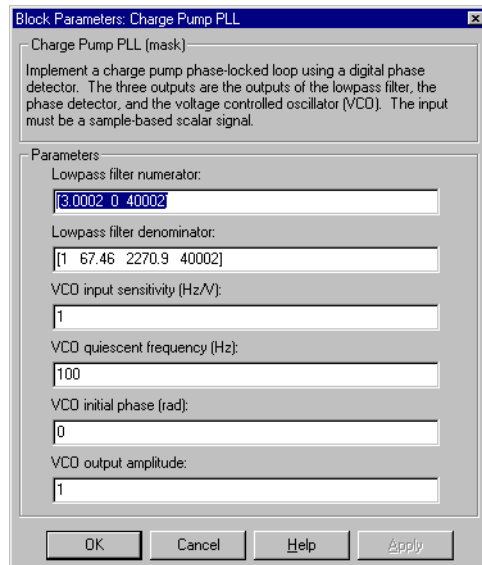
- A voltage-controlled oscillator (VCO). You specify characteristics of the VCO using the **VCO input sensitivity**, **VCO quiescent frequency**, **VCO initial phase**, and **VCO output amplitude** parameters.

The input signal represents the received signal. The input must be a sample-based scalar signal. The three output ports produce:

- The output of the filter
- The output of the phase detector
- The output of the VCO

A sequential logic phase detector operates on the zero crossings of the signal waveform. The equilibrium point of the phase difference between the input signal and the VCO signal equals π . The sequential logic detector can compensate for any frequency difference that might exist between a VCO and an incoming signal frequency. Hence, the sequential logic phase detector acts as a frequency detector.

Dialog Box



Lowpass filter numerator

The numerator of the lowpass filter's transfer function, represented as a vector that lists the coefficients in order of descending powers of s .

Lowpass filter denominator

The denominator of the lowpass filter's transfer function, represented as a vector that lists the coefficients in order of descending powers of s .

VCO input sensitivity (Hz/V)

This value scales the input to the VCO and, consequently, the shift from the **VCO quiescent frequency** value. The units of **VCO input sensitivity** are Hertz per volt.

VCO quiescent frequency (Hz)

The frequency of the VCO signal when the voltage applied to it is zero. This should match the frequency of the input signal.

VCO initial phase (rad)

The initial phase of the VCO signal.

Charge Pump PLL

VCO output amplitude

The amplitude of the VCO signal.

See Also

Phase-Locked Loop

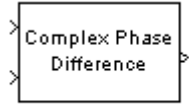
References

For more information about digital phase-locked loops, see the works listed in “Selected Bibliography for Synchronization” on page 2-92.

Purpose Output the phase difference between the two complex input signals

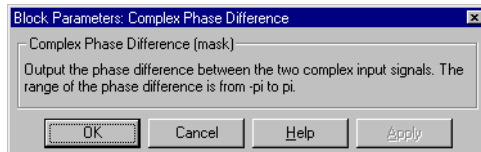
Library Sequence Operations, in Basic Comm Functions

Description The Complex Phase Difference block accepts two complex input signals that have the same size and frame status. The output is the phase difference from the second to the first, measured in radians. The elements of the output are between $-\pi$ and π .



The input signals can have any size or frame status. This block processes each pair of elements independently.

Dialog Box



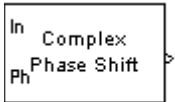
See Also Complex Phase Shift

Complex Phase Shift

Purpose Shift the phase of the complex input signal by the second input value

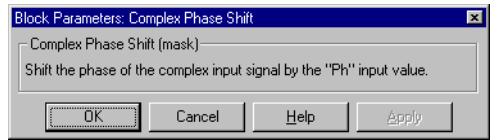
Library Sequence Operations, in Basic Comm Functions

Description The Complex Phase Shift block accepts a complex signal at the port labeled *I n*. The output is the result of shifting this signal's phase by an amount specified by the real signal at the input port labeled *Ph*. The *Ph* input is measured in radians, and must have the same size and frame status as the *I n* input.



The input signals can have any size or frame status. This block processes each pair of corresponding elements independently.

Dialog Box



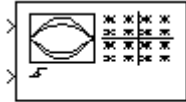
See Also Complex Phase Difference

Continuous-Time Eye and Scatter Diagrams

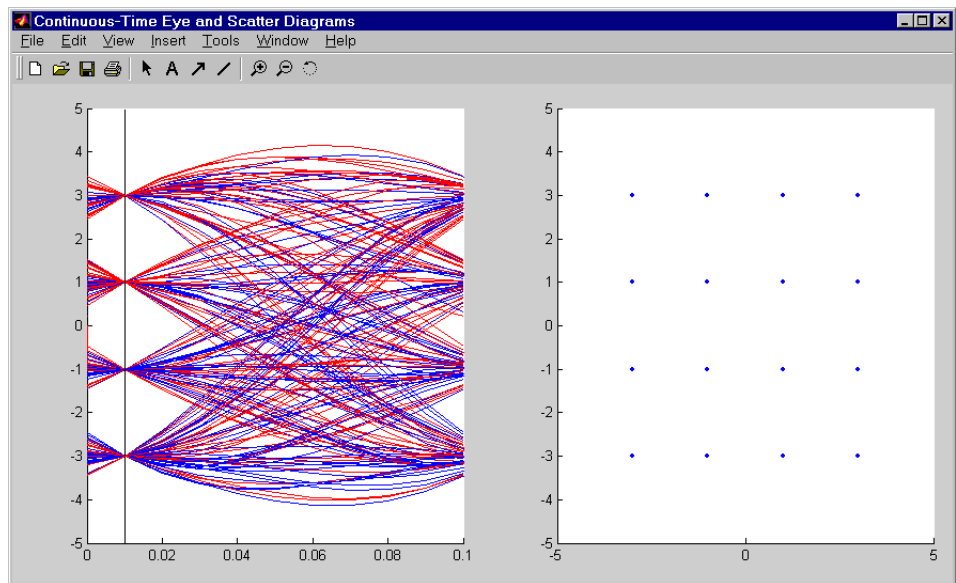
Purpose Produce eye diagram, scatter, or x-y plots, using trigger to set decision timing

Library Comm Sinks

Description



The Continuous-Time Eye and Scatter Diagrams block plots eye diagrams, scatter diagrams, and X-Y diagrams from a continuous-time input signal. The **Diagram type** parameter determines which plots the block produces. The block draws the diagrams in a single window, as in the case of the eye diagram and scatter diagram below.



The first input is a complex message signal. It must be a sample-based scalar signal. The eye diagram and the X-Y diagram both record the trajectories of the message signal in continuous time.

The second input is a scalar trigger signal that determines the decision timing for the scatter diagram. At each rising edge of the trigger signal, the block plots a vertical line in the eye diagram and adds a new point to the scatter plot.

Continuous-Time Eye and Scatter Diagrams

The **Trace period** parameter is the number of seconds represented by the horizontal axis in the eye diagram. The **Trace offset** parameter is the time value at the left edge of the horizontal axis of the eye diagram.

Tip This block works better if the model's simulation step size is suitable for your data. From the model window's **Simulation** menu, choose **Simulation parameters** and then choose a value for the **Max step size** parameter. Try multiplying this block's **Trace period** parameter by 1/8 or 1/16.

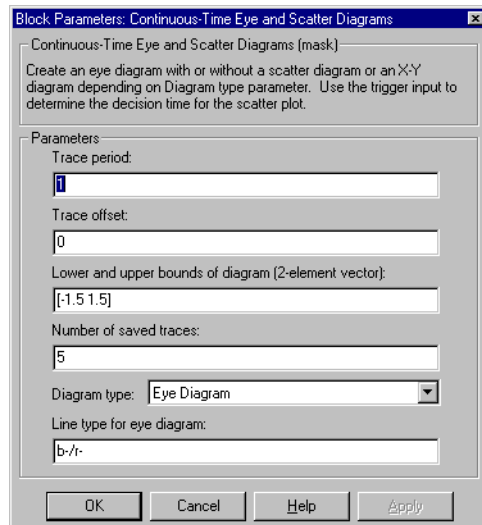
To specify the plotting color as well as the line type and/or marker type, use the **Line type** parameters that appear after you select a value for the **Diagram type** parameter. In the **Line type for eye diagram** parameter, use a slash (/) to separate the specifications for the real and imaginary components of the input signal. Choices for the color, marker, and line types are in the table below.

Color Characters		Marker-Type Characters		Line-Type Characters	
y	Yellow	.	Point	-	Solid
m	Magenta	o	Circle	:	Dotted
c	Cyan	x	Cross	- .	Dash-dot
r	Red	+	Plus sign	- -	Dashed
g	Green	*	Asterisk		
b	Blue	s	Square		
w	White	d	Diamond		
k	Black	v	Triangle (down)		
		^	Triangle (up)		
		<	Triangle (left)		
		>	Triangle (right)		

Continuous-Time Eye and Scatter Diagrams

Color Characters		Marker-Type Characters		Line-Type Characters	
		p	Five-pointed star		
		h	Six-pointed star		

Dialog Box



Trace period

The duration of the horizontal axis of the eye diagram, in seconds.

Trace offset

The time at the leftmost edge of the horizontal axis of the eye diagram.

Lower and upper bounds of diagram

A two-element vector containing the minimum and maximum signal values in the diagrams.

Number of saved traces

The number of curves in the eye diagram, or points in the scatter plot, that are visible after you resize or restore the figure window.

Diagram type

The diagram(s) that the block produces.

Continuous-Time Eye and Scatter Diagrams

Line type for eye diagram

A string that specifies the color and the line type for the eye diagram. This field appears only when the **Diagram type** parameter is set to an option that includes an eye diagram.

Line type for scatter diagram

A string that specifies the color and the marker type for the scatter diagram. This field appears only when the **Diagram type** parameter is set to an option that includes a scatter diagram.

Line type for X-Y diagram

A string that specifies the color and the line type for the X-Y diagram. This field appears only when the **Diagram type** parameter is set to an option that includes an X-Y diagram.

Limitations

Since this block uses an M-file, you cannot generate C code for it using the Real-Time Workshop.

See Also

Discrete-Time Eye and Scatter Diagrams

Purpose Restore ordering of symbols that were permuted using shift registers

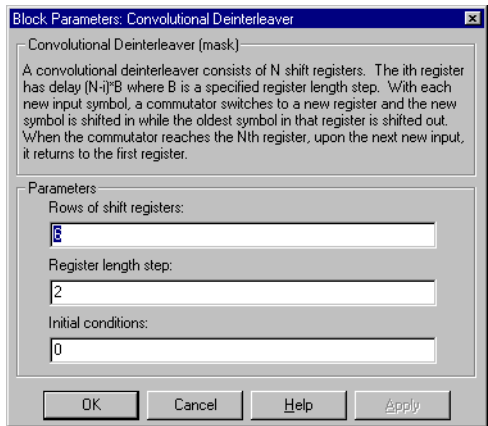
Library Convolutional sublibrary of Interleaving

Description The Convolutional Deinterleaver block recovers a signal that was interleaved using the Convolutional Interleaver block. The parameters in the two blocks should have the same values.



The input can be either a scalar or a frame-based column vector. It can be real or complex. The sample times of the input and output signals are the same.

Dialog Box



Rows of shift registers

The number of shift registers that the block uses internally.

Register length step

The difference in symbol capacity of each successive shift register, where the last register holds zero symbols.

Initial conditions

The values that fill each shift register when the simulation begins.

Pair Block Convolutional Interleaver

See Also General Multiplexed Deinterleaver, Helical Deinterleaver

Convolutional Deinterleaver

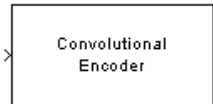
References

- [1] Clark, George C. Jr. and J. Bibb Cain. *Error-Correction Coding for Digital Communications*. New York: Plenum Press, 1981.
- [2] Forney, G., D., Jr. "Burst-Correcting Codes for the Classic Bursty Channel." *IEEE Transactions on Communications*, vol. COM-19, October 1971. 772-781.
- [3] Ramsey, J. L. "Realization of Optimum Interleavers." *IEEE Transactions on Information Theory*, IT-16 (3), May 1970. 338-345.

Purpose Create a convolutional code from binary data

Library Convolutional sublibrary of Channel Coding

Description The Convolutional Encoder block encodes a sequence of binary input vectors to produce a sequence of binary output vectors. This block can process multiple symbols at a time.



Input and Output Sizes

If the encoder takes k input bit streams (that is, can receive 2^k possible input symbols), then this block's input vector length is $L \cdot k$ for some positive integer L . Similarly, if the encoder produces n output bit streams (that is, can produce 2^n possible output symbols), then this block's output vector length is $L \cdot n$.

The input can be a sample-based vector with $L = 1$, or a frame-based column vector with any positive integer for L .

Specifying the Encoder

To define the convolutional encoder, use the **Trellis structure** parameter. This parameter is a MATLAB structure whose format is described in the section, "Trellis Description of a Convolutional Encoder," in the *Communications Toolbox User's Guide*. You can use this parameter field in two ways:

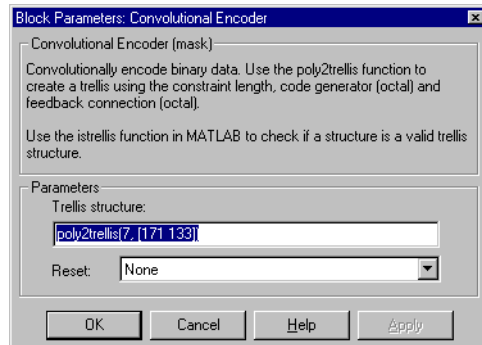
- If you have a variable in the MATLAB workspace that contains the trellis structure, then enter its name as the **Trellis structure** parameter. This way is preferable because it causes Simulink to spend less time updating the diagram at the beginning of each simulation, compared to the usage in the next bulleted item.
- If you want to specify the encoder using its constraint length, generator polynomials, and possibly feedback connection polynomials, then use a `poly2trellis` command within the **Trellis structure** field. For example, to use an encoder with a constraint length of 7, code generator polynomials of 171 and 133 (in octal numbers), and a feedback connection of 171 (in octal), set the **Trellis structure** parameter to
`poly2trellis(7, [171 133], 171)`

The encoder registers begin in the all-zeros state. You can configure the encoder so that it resets its registers to the all-zeros state during the course of the simulation. To do this, use one of these values of the **Reset** parameter:

Convolutional Encoder

- The value **None** indicates that the encoder never resets.
- The value **On each frame** indicates that the encoder resets at the beginning of each frame, before processing the next frame of input data
- The value **On nonzero Rst input** causes the block to have a second input port, labeled Rst. The signal at the Rst port is a scalar signal. When it is nonzero, the encoder resets before processing the data at the first input port.

Dialog Box



Trellis structure

MATLAB structure that contains the trellis description of the convolutional encoder.

Reset

Determines whether and under what circumstances the encoder resets to the all-zeros state before processing the input data. Choices are **None**, **On each frame**, and **On nonzero Rst input**. The last option causes the block to have a second input port, labeled Rst.

See Also

Viterbi Decoder, APP Decoder

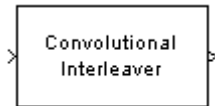
References

- [1] Clark, George C. Jr. and J. Bibb Cain. *Error-Correction Coding for Digital Communications*. New York: Plenum Press, 1981.
- [2] Gitlin, Richard D., Jeremiah F. Hayes, and Stephen B. Weinstein. *Data Communications Principles*. New York: Plenum, 1992.

Purpose Permute input symbols using a set of shift registers

Library Convolutional sublibrary of Interleaving

Description

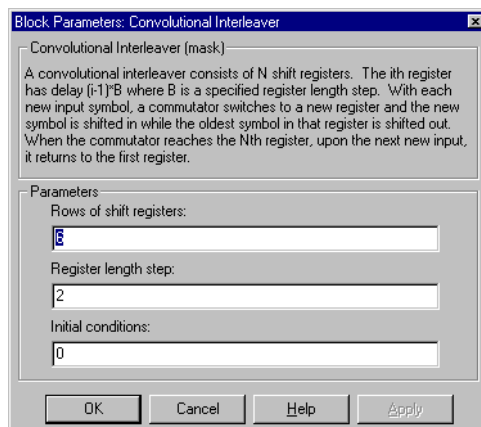


The Convolutional Interleaver block permutes the symbols in the input signal. Internally, it uses a set of shift registers. The delay value of the k th shift register is $(k-1)$ times the **Register length step** parameter. The number of shift registers is the value of the **Rows of shift registers** parameter.

The **Initial conditions** parameter indicates the values that fill each shift register at the beginning of the simulation (except for the first shift register, which has zero delay). If **Initial conditions** is a scalar, then its value fills all shift registers except the first; if **Initial conditions** is a column vector whose length is the **Rows of shift registers** parameter, then each entry fills the corresponding shift register. The value of the first element of the **Initial conditions** parameter is unimportant, since the first shift register has zero delay.

The input can be either a scalar or a frame-based column vector. It can be real or complex. The sample times of the input and output signals are the same.

Dialog Box



Rows of shift registers

The number of shift registers that the block uses internally.

Convolutional Interleaver

Register length step

The number of additional symbols that fit in each successive shift register, where the first register holds zero symbols.

Initial conditions

The values that fill each shift register when the simulation begins.

Pair Block Convolutional Deinterleaver

See Also General Multiplexed Interleaver, Helical Interleaver

References

- [1] Clark, George C. Jr. and J. Bibb Cain. *Error-Correction Coding for Digital Communications*. New York: Plenum Press, 1981.
- [2] Forney, G., D., Jr. "Burst-Correcting Codes for the Classic Bursty Channel." *IEEE Transactions on Communications*, vol. COM-19, October 1971. 772-781.
- [3] Ramsey, J. L. "Realization of Optimum Interleavers." *IEEE Transactions on Information Theory*, IT-16 (3), May 1970. 338-345.

Purpose Demodulate CPFSK-modulated data

Library CPM, in Digital Baseband sublibrary of Modulation

Description



The CPFSK Demodulator Baseband block demodulates a signal that was modulated using the continuous phase frequency shift keying method. The input is a baseband representation of the modulated signal. The **M-ary number** parameter, M , is the size of the input alphabet. M must have the form 2^K for some positive integer K .

The **Modulation index** parameter times π radians is the phase shift in the modulated signal due to the latest symbol, when that symbol is the integer 1. The **Phase offset** parameter is the initial phase of the modulated waveform.

Traceback Length and Output Delays

Internally, this block creates a trellis description of the modulation scheme and uses the Viterbi algorithm. The **Traceback length** parameter, D , in this block is the number of trellis branches used to construct each traceback path. D influences the output delay, which is the number of zero symbols that precede the first meaningful demodulated value in the output.

- If the input signal is sample-based, then the delay consists of $D+1$ zero symbols.
- If the input signal is frame-based, then the delay consists of D zero symbols.

Outputs and Symbol Sets

If the **Output type** parameter is set to **Integer**, then the block produces odd integers between $-(M-1)$ and $M-1$.

If the **Output type** parameter is set to **Bit**, then the block produces groupings of K bits. Each grouping is called a binary *word*.

In binary output mode, the block first maps each input symbol to an intermediate value as in the integer output mode. The block then maps the odd integer k to the nonnegative integer $(k+M-1)/2$. Finally, the block maps each nonnegative integer to a binary word, using a mapping that depends on whether the **Symbol set ordering** parameter is set to **Binary** or **Gray**. For more information about Gray and binary coding, see “Binary-Valued and Integer-Valued Signals” on page 2-68.

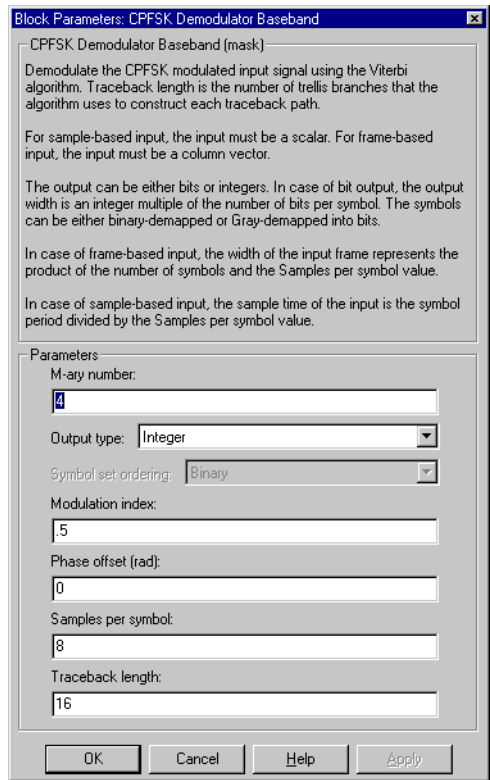
CPFSK Demodulator Baseband

The input can be either a scalar or a frame-based column vector.

Processing an Upsampled Modulated Signal

The input signal can be an upsampled version of the modulated signal. The **Samples per symbol** parameter is the upsampling factor. It must be a positive integer. For more information, see “Upsampled Signals and Rate Changes” on page 2-72.

Dialog Box



M-ary number

The size of the alphabet.

Output type

Determines whether the output consists of integers or groups of bits.

Symbol set ordering

Determines how the block maps each integer to a group of output bits. This field is active only when **Output type** is set to **Bit**.

Modulation index

The number of half-revolutions of phase shift in the modulated signal after modulating the latest symbol of 1.

Phase offset (rad)

The initial phase of the modulated waveform.

Samples per symbol

The number of input samples that represent each modulated symbol.

Traceback length

The number of trellis branches that the Viterbi Decoder block uses to construct each traceback path.

Pair Block

CPFSK Modulator Baseband

See Also

CPM Demodulator Baseband, Viterbi Decoder, M-FSK Demodulator Baseband

References

[1] Anderson, John B., Tor Aulin, and Carl-Erik Sundberg. *Digital Phase Modulation*. New York: Plenum Press, 1986.

CPFSK Demodulator Passband

Purpose Demodulate CPFSK-modulated data

Library CPM, in Digital Passband sublibrary of Modulation

Description



The CPFSK Demodulator Passband block demodulates a signal that was modulated using the continuous phase frequency shift keying method. The input is a passband representation of the modulated signal. The **M-ary number** parameter, M, is the size of the input alphabet. M must have the form 2^K for some positive integer K.

This block converts the input to an equivalent baseband representation and then uses the baseband equivalent block, CPFSK Demodulator Baseband, for internal computations. The following parameters in this block are the same as those of the baseband equivalent block:

- **M-ary number**
- **Output type**
- **Signal set ordering**
- **Modulation index**
- **Traceback length**

The input must be a sample-based scalar signal.

Parameters Specific to Passband Simulation

Passband simulation uses a carrier signal. The **Carrier frequency** and **Carrier initial phase** parameters specify the frequency and initial phase, respectively, of the carrier signal. The **Input sample time** parameter specifies the sample time of the input signal, while the **Symbol period** parameter equals the sample time of the output signal.

This block uses a baseband representation of the modulated signal as an intermediate signal during internal computations. The **Baseband samples per symbol** parameter indicates how many baseband samples correspond to each integer or binary word in the output.

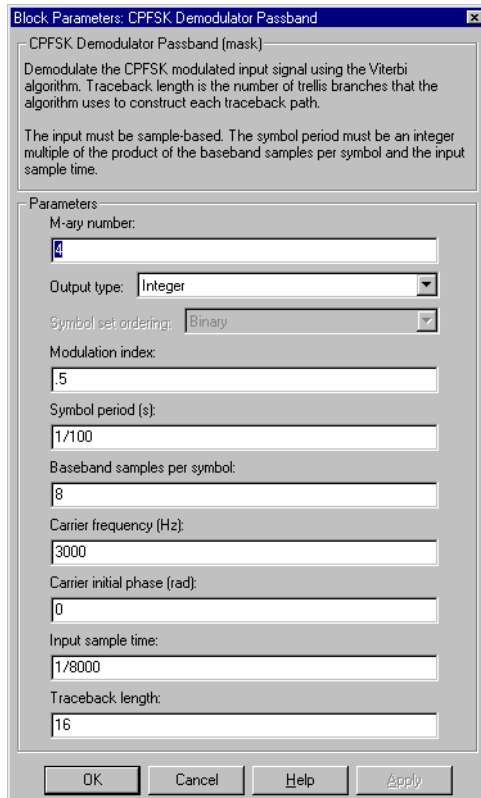
The timing-related parameters must satisfy these relationships:

- **Input sample time** > (**Carrier frequency**)⁻¹
- **Symbol period** < [2***Carrier frequency** + 2/(**Input sample time**)]⁻¹

- **Input sample time** = $K \times \text{Symbol period} \times \text{Baseband samples per symbol}$ for some integer K

Also, this block incurs an extra output period of delay compared to its baseband equivalent block.

Dialog Box



The dialog box is titled "Block Parameters: CPFSK Demodulator Passband". It contains a description of the block's function and a list of parameters to be configured.

CPFSK Demodulator Passband (mask)
Demodulate the CPFSK modulated input signal using the Viterbi algorithm. Traceback length is the number of trellis branches that the algorithm uses to construct each traceback path.
The input must be sample-based. The symbol period must be an integer multiple of the product of the baseband samples per symbol and the input sample time.

Parameters

M-ary number:	4
Output type:	Integer
Symbol set ordering:	Binary
Modulation index:	5
Symbol period (s):	1/100
Baseband samples per symbol:	8
Carrier frequency (Hz):	3000
Carrier initial phase (rad):	0
Input sample time:	1/8000
Traceback length:	16

Buttons: OK, Cancel, Help, Apply

M-ary number

The size of the alphabet.

Output type

Determines whether the output consists of integers or groups of bits.

CPFSK Demodulator Passband

Symbol set ordering

Determines how the block maps each integer to a group of output bits. This field is active only when **Output type** is set to **Bit**.

Modulation index

The number of half-revolutions of phase shift in the modulated signal after modulating the latest symbol of 1.

Symbol period (s)

The symbol period, which equals the sample time of the output.

Baseband samples per symbol

The number of baseband samples that represent each modulated symbol, after the block converts the passband input to a baseband intermediary signal.

Carrier frequency (Hz)

The frequency of the carrier.

Carrier initial phase (rad)

The initial phase of the carrier in radians.

Input sample time

The sample time of the input signal.

Traceback length

The number of trellis branches that the Viterbi Decoder block uses to construct each traceback path.

Pair Block CPFSK Modulator Passband

See Also CPFSK Demodulator Baseband, Viterbi Decoder, M-FSK Demodulator Passband

References [1] Anderson, John B., Tor Aulin, and Carl-Erik Sundberg. *Digital Phase Modulation*. New York: Plenum Press, 1986.

Purpose Modulate using the continuous phase frequency shift keying method

Library CPM, in Digital Baseband sublibrary of Modulation

Description



The CPFSK Modulator Baseband block modulates using the continuous phase frequency shift keying method. The output is a baseband representation of the modulated signal. The **M-ary number** parameter, M , is the size of the input alphabet. M must have the form 2^K for some positive integer K .

The **Modulation index** parameter times π radians is the phase shift due to the latest symbol when that symbol is the integer 1. The **Phase offset** parameter is the initial phase of the output waveform, measured in radians.

For the exact definitions of the rectangular pulse shape that this block uses, see the work by Anderson, Aulin, and Sundberg listed in “References” on page 4-117.

Inputs and Symbol Sets

If the **Input type** parameter is set to **Integer**, then the block accepts odd integers between $-(M-1)$ and $M-1$.

If the **Input type** parameter is set to **Bit**, then the block accepts groupings of K bits. Each grouping is called a binary *word*. The input vector length must be an integer multiple of K .

In binary input mode, the block maps each binary word to an integer between 0 and $M-1$, using a mapping that depends on whether the **Symbol set ordering** parameter is set to **Binary** or **Gray**. The block then maps the integer k to the intermediate value $2k-(M-1)$ and proceeds as in the integer input mode. For more information, see “Binary-Valued and Integer-Valued Signals” on page 2-68.

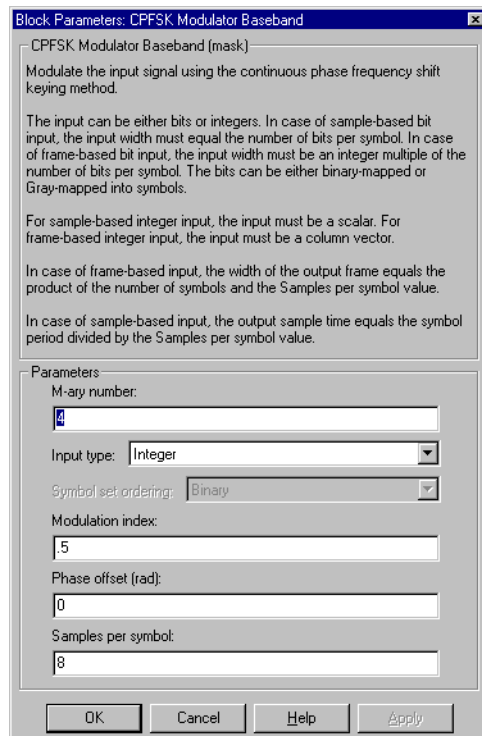
The input can be either a scalar or a frame-based column vector. If **Input type** is **Bit**, then the input can also be a vector of length K .

Upsampling the Modulated Signal

This block can output an upsampled version of the modulated signal. The **Samples per symbol** parameter is the upsampling factor. It must be a positive integer. For more information, see “Upsampled Signals and Rate Changes” on page 2-72.

CPFSK Modulator Baseband

Dialog Box



M-ary number

The size of the alphabet.

Input type

Indicates whether the input consists of integers or groups of bits.

Symbol set ordering

Determines how the block maps each group of input bits to a corresponding integer. This field is active only when **Input type** is set to **Bit**.

Modulation index

The number of half-revolutions of phase shift due to the latest symbol when that symbol is the integer 1.

Phase offset (rad)

The initial phase of the output waveform.

Samples per symbol

The number of output samples that the block produces for each integer or binary word in the input.

Pair Block CPFSK Demodulator Baseband

See Also CPM Modulator Baseband, M-FSK Modulator Baseband

References [1] Anderson, John B., Tor Aulin, and Carl-Erik Sundberg. *Digital Phase Modulation*. New York: Plenum Press, 1986.

CPFSK Modulator Passband

Purpose Modulate using the continuous phase frequency shift keying method

Library CPM, in Digital Passband sublibrary of Modulation

Description



The CPFSK Modulator Passband block modulates using the continuous phase frequency shift keying method. The output is a passband representation of the modulated signal. The **M-ary number** parameter, M , is the size of the input alphabet. M must have the form 2^K for some positive integer K .

This block uses the baseband equivalent block, CPFSK Modulator Baseband, for internal computations and converts the resulting baseband signal to a passband representation. The following parameters in this block are the same as those of the baseband equivalent block:

- **M-ary number**
- **Input type**
- **Symbol set ordering**
- **Modulation index**

The input must be sample-based. If the **Input type** parameter is **Bit**, then the input must be a vector of length $\log_2(M)$. If the **Input type** parameter is **Integer**, then the input must be a scalar.

Parameters Specific to Passband Simulation

Passband simulation uses a carrier signal. The **Carrier frequency** and **Carrier initial phase** parameters specify the frequency and initial phase, respectively, of the carrier signal. The **Symbol period** parameter must equal the sample time of the input signal, while the **Output sample time** parameter determines the sample time of the output signal.

This block uses a baseband representation of the modulated signal as an intermediate result during internal computations. The **Baseband samples per symbol** parameter indicates how many baseband samples correspond to each integer or binary word in the input, before the block converts them to a passband output.

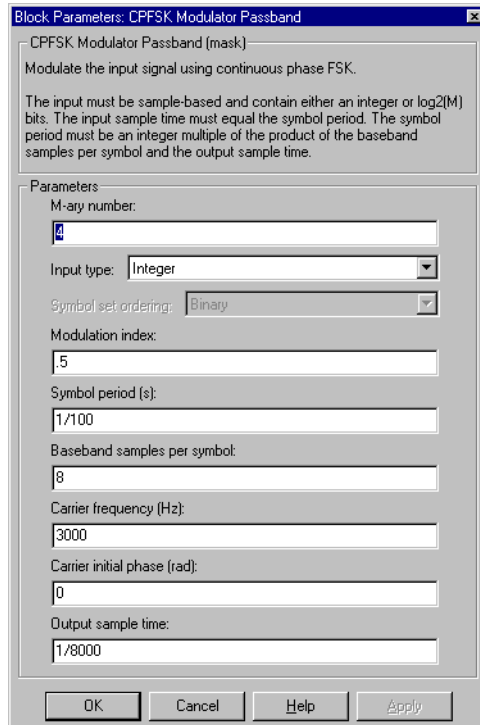
The timing-related parameters must satisfy these relationships:

- **Symbol period** $> (\text{Carrier frequency})^{-1}$
- **Output sample time** $< [2 * \text{Carrier frequency} + 2/(\text{Symbol period})]^{-1}$

- **Symbol period** = $K \times \text{Output sample time} \times \text{Baseband samples per symbol}$ for some integer K

Furthermore, **Carrier frequency** is typically much larger than the highest frequency of the unmodulated signal.

Dialog Box



The dialog box is titled "Block Parameters: CPFSK Modulator Passband". It contains a description of the block's function and a list of parameters to be configured.

CPFSK Modulator Passband (mask)
Modulate the input signal using continuous phase FSK.

The input must be sample-based and contain either an integer or log2(M) bits. The input sample time must equal the symbol period. The symbol period must be an integer multiple of the product of the baseband samples per symbol and the output sample time.

Parameters:

- M-ary number: 4
- Input type: Integer
- Symbol set ordering: Binary
- Modulation index: 5
- Symbol period (s): 1/100
- Baseband samples per symbol: 8
- Carrier frequency (Hz): 3000
- Carrier initial phase (rad): 0
- Output sample time: 1/8000

Buttons: OK, Cancel, Help, Apply

M-ary number

The size of the alphabet.

Input type

Indicates whether the input consists of integers or groups of bits.

Symbol set ordering

Determines how the block maps each group of input bits to a corresponding integer. This field is active only when **Input type** is set to **Bit**.

CPFSK Modulator Passband

Modulation index

The number of half-revolutions of phase shift due to the latest symbol when that symbol is the integer 1.

Symbol period (s)

The symbol period, which must equal the sample time of the input.

Baseband samples per symbol

The number of baseband samples that correspond to each integer or binary word in the input, before the block converts them to a passband output.

Carrier frequency (Hz)

The frequency of the carrier.

Carrier initial phase (rad)

The initial phase of the carrier in radians.

Output sample time

The sample time of the output signal.

Pair Block

CPFSK Demodulator Passband

See Also

CPFSK Modulator Baseband, M-FSK Modulator Passband

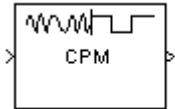
References

[1] Anderson, John B., Tor Aulin, and Carl-Erik Sundberg. *Digital Phase Modulation*. New York: Plenum Press, 1986.

Purpose Demodulate CPM-modulated data

Library CPM, in Digital Baseband sublibrary of Modulation

Description



The CPM Demodulator Baseband block demodulates a signal that was modulated using continuous phase modulation. The input is a baseband representation of the modulated signal. The **M-ary number** parameter, M , is the size of the input alphabet. M must have the form 2^K for some positive integer K .

The input can be either a scalar or a frame-based column vector.

The **Modulation index**, **Frequency pulse shape**, **Rolloff**, **BT product**, **Pulse length**, **Symbol prehistory**, and **Phase offset** parameters are as described on the reference page for the CPM Modulator Baseband block.

Traceback Length and Output Delays

Internally, this block creates a trellis description of the modulation scheme and uses the Viterbi algorithm. The **Traceback length** parameter, D , in this block is the number of trellis branches used to construct each traceback path. D influences the output delay, which is the number of zero symbols that precede the first meaningful demodulated value in the output.

- If the input signal is sample-based, then the delay consists of $D+1$ zero symbols.
- If the input signal is frame-based, then the delay consists of D zero symbols.

Outputs and Symbol Sets

If the **Output type** parameter is set to **Integer**, then the block produces odd integers between $-(M-1)$ and $M-1$.

If the **Output type** parameter is set to **Bit**, then the block produces groupings of K bits. Each grouping is called a binary *word*.

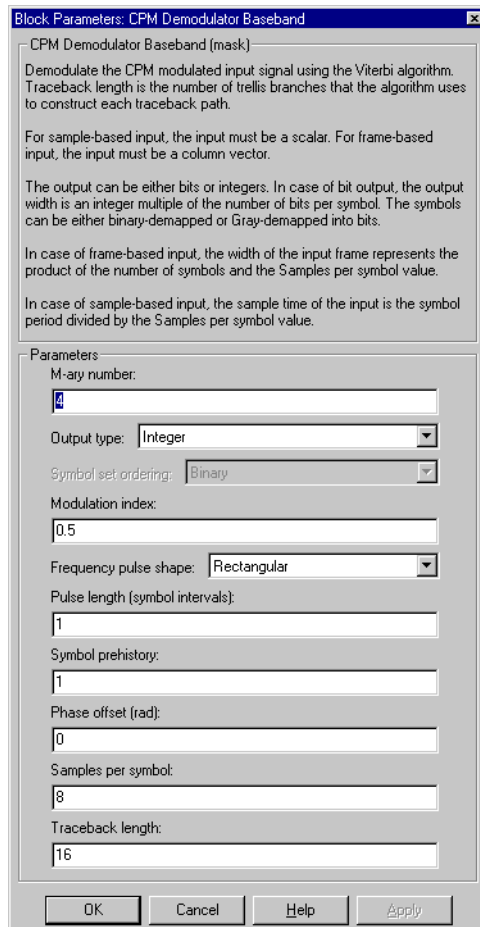
In binary output mode, the block first maps each input symbol to an intermediate value as in the integer output mode. The block then maps the odd integer k to the nonnegative integer $(k+M-1)/2$. Finally, the block maps each nonnegative integer to a binary word, using a mapping that depends on whether the **Symbol set ordering** parameter is set to **Binary** or **Gray**. For

more information about Gray and binary coding, see “Binary-Valued and Integer-Valued Signals” on page 2-68.

Processing an Upsampled Modulated Signal

The input signal can be an upsampled version of the modulated signal. The **Samples per symbol** parameter is the upsampling factor. It must be a positive integer. For more information, see “Upsampled Signals and Rate Changes” on page 2-72.

Dialog Box



The dialog box is titled "Block Parameters: CPM Demodulator Baseband". It contains a text area with the following text:

CPM Demodulator Baseband (mask)

Demodulate the CPM modulated input signal using the Viterbi algorithm. Traceback length is the number of trellis branches that the algorithm uses to construct each traceback path.

For sample-based input, the input must be a scalar. For frame-based input, the input must be a column vector.

The output can be either bits or integers. In case of bit output, the output width is an integer multiple of the number of bits per symbol. The symbols can be either binary-demapped or Gray-demapped into bits.

In case of frame-based input, the width of the input frame represents the product of the number of symbols and the Samples per symbol value.

In case of sample-based input, the sample time of the input is the symbol period divided by the Samples per symbol value.

Parameters

M-ary number:

Output type:

Symbol set ordering:

Modulation index:

Frequency pulse shape:

Pulse length (symbol intervals):

Symbol prehistory:

Phase offset (rad):

Samples per symbol:

Traceback length:

Buttons: OK, Cancel, Help, Apply

M-ary number

The size of the alphabet.

Output type

Determines whether the output consists of integers or groups of bits.

CPM Demodulator Baseband

Symbol set ordering

Determines how the block maps each integer to a group of output bits. This field is active only when **Output type** is set to **Bit**.

Modulation index

The number of half-revolutions of phase shift in the modulated signal after modulating the latest symbol of 1.

Frequency pulse shape

The type of pulse shaping that the corresponding modulator uses to smooth the phase transitions of the modulated signal.

Rolloff

The rolloff factor of the raised cosine filter. This field appears only when **Frequency pulse shape** is set to **Spectral Raised Cosine**.

BT product

The product of bandwidth and time. This field appears only when **Frequency pulse shape** is set to **Gaussian**.

Pulse length (symbol intervals)

The length of the frequency pulse shape.

Symbol prehistory

The data symbols used by the modulator before the start of the simulation.

Phase offset (rad)

The initial phase of the modulated waveform.

Samples per symbol

The number of input samples that represent each modulated symbol.

Traceback length

The number of trellis branches that the Viterbi Decoder block uses to construct each traceback path.

Pair Block

CPM Modulator Baseband

See Also

CPFSK Demodulator Baseband, GMSK Demodulator Baseband, MSK Demodulator Baseband, Viterbi Decoder

References

[1] Anderson, John B., Tor Aulin, and Carl-Erik Sundberg. *Digital Phase Modulation*. New York: Plenum Press, 1986.

Purpose Demodulate CPM-modulated data

Library CPM, in Digital Passband sublibrary of Modulation

Description



The CPM Demodulator Passband block demodulates a signal that was modulated using continuous phase modulation. The input is a passband representation of the modulated signal. The **M-ary number** parameter, M , is the size of the input alphabet. M must have the form 2^K for some positive integer K .

This block converts the input to an equivalent baseband representation and then uses the baseband equivalent block, CPM Demodulator Baseband, for internal computations. The following parameters in this block are the same as those of the baseband equivalent block:

- **M-ary number**
- **Output type**
- **Symbol set ordering**
- **Modulation index**
- **Frequency pulse shape**
- **Rolloff**
- **BT product**
- **Pulse length**
- **Symbol prehistory**
- **Traceback length**

The input must be a sample-based scalar signal.

Parameters Specific to Passband Simulation

Passband simulation uses a carrier signal. The **Carrier frequency** and **Carrier initial phase** parameters specify the frequency and initial phase, respectively, of the carrier signal. The **Input sample time** parameter specifies the sample time of the input signal, while the **Symbol period** parameter equals the sample time of the output signal.

This block uses a baseband representation of the modulated signal as an intermediate signal during internal computations. The **Baseband samples per symbol** parameter indicates how many baseband samples correspond to each integer or binary word in the output.

CPM Demodulator Passband

The timing-related parameters must satisfy these relationships:

- **Input sample time** > **(Carrier frequency)**⁻¹
- **Symbol period** < **[2*Carrier frequency + 2/(Input sample time)]**⁻¹
- **Input sample time** = **K*Symbol period*Baseband samples per symbol** for some integer K

Also, this block incurs an extra output period of delay compared to its baseband equivalent block.

Dialog Box

Block Parameters: CPM Demodulator Passband

CPM Demodulator Passband (mask)

Demodulate the CPM modulated input signal using the Viterbi algorithm. Traceback length is the number of trellis branches that the algorithm uses to construct each traceback path.

The input must be sample-based. The symbol period must be an integer multiple of the product of the baseband samples per symbol and the input sample time.

The Symbol prehistory parameter is the data symbol(s) used before the start of the simulation.

Parameters

M-ary number:

Output type:

Symbol set ordering:

Modulation index:

Frequency pulse shape:

Pulse length (symbol intervals):

Symbol prehistory:

Symbol period (s):

Baseband samples per symbol:

Carrier frequency (Hz):

Carrier initial phase (rad):

Input sample time:

Traceback length:

OK Cancel Help Apply

M-ary number

The size of the alphabet.

Output type

Determines whether the output consists of integers or groups of bits.

Symbol set ordering

Determines how the block maps each integer to a group of output bits. This field is active only when **Output type** is set to **Bit**.

Modulation index

The number of half-revolutions of phase shift in the modulated signal after modulating the latest symbol of 1.

Frequency pulse shape

The type of pulse shaping that the corresponding modulator uses to smooth the phase transitions of the modulated signal.

Rolloff

The rolloff factor of the raised cosine filter. This field appears only when **Frequency pulse shape** is set to **Spectral Raised Cosine**.

BT product

The product of bandwidth and time. This field appears only when **Frequency pulse shape** is set to **Gaussian**.

Pulse length (symbol intervals)

The length of the frequency pulse shape.

Symbol prehistory

The data symbols used by the modulator before the start of the simulation.

Symbol period (s)

The symbol period, which equals the sample time of the output.

Baseband samples per symbol

The number of baseband samples that represent each modulated symbol, after the block converts the passband input to a baseband intermediary signal.

Carrier frequency (Hz)

The frequency of the carrier.

Carrier initial phase (rad)

The initial phase of the carrier in radians.

Input sample time

The sample time of the input signal.

Traceback length

The number of trellis branches that the Viterbi Decoder block uses to construct each traceback path.

Pair Block

CPM Modulator Passband

See Also

CPM Demodulator Baseband, Viterbi Decoder

References

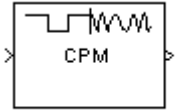
[1] Anderson, John B., Tor Aulin, and Carl-Erik Sundberg. *Digital Phase Modulation*. New York: Plenum Press, 1986.

CPM Modulator Baseband

Purpose Modulate using continuous phase modulation

Library CPM, in Digital Baseband sublibrary of Modulation

Description



The CPM Modulator Baseband block modulates using continuous phase modulation. The output is a baseband representation of the modulated signal. The **M-ary number** parameter, M , is the size of the input alphabet. M must have the form 2^K for some positive integer K .

Continuous phase modulation uses pulse shaping to smooth the phase transitions of the modulated signal. Using the **Frequency pulse shape** parameter, you can choose these types of pulse shapes:

- **Rectangular**
- **Raised Cosine**
- **Spectral Raised Cosine**

This option requires an additional parameter, **Rolloff**. The **Rolloff** parameter, which affects the spectrum of the pulse, is a scalar between zero and one.

- **Gaussian**

This option requires an additional parameter, **BT product**. The **BT product** parameter, which represents bandwidth multiplied by time, is a nonnegative scalar. It is used to reduce the bandwidth at the expense of increased intersymbol interference.

- **Tamed FM** (tamed frequency modulation)

For the exact definitions of these pulse shapes, see the work by Anderson, Aulin, and Sundberg listed in “References” on page 4-133. Each pulse shape has a corresponding pulse duration. The **Pulse length** parameter measures this quantity in symbol intervals.

The **Modulation index** parameter times π radians is the phase shift due to the latest symbol when that symbol is the integer 1. The **Phase offset** parameter is the initial phase of the output waveform, measured in radians.

The **Symbol prehistory** parameter is a scalar or vector that specifies the data symbols used before the start of the simulation, in reverse chronological order. If it is a vector, then its length must be one less than the **Pulse length** parameter.

Inputs and Symbol Sets

If the **Input type** parameter is set to **Integer**, then the block accepts odd integers between $-(M-1)$ and $M-1$.

If the **Input type** parameter is set to **Bit**, then the block accepts groupings of K bits. Each grouping is called a binary *word*. The input vector length must be an integer multiple of K .

In binary input mode, the block maps each binary word to an integer between 0 and $M-1$, using a mapping that depends on whether the **Symbol set ordering** parameter is set to **Binary** or **Gray**. The block then maps the integer k to the intermediate value $2^{k-(M-1)}$ and proceeds as in the integer input mode. For more information, see “Binary-Valued and Integer-Valued Signals” on page 2-68.

The input can be either a scalar or a frame-based column vector. If **Input type** is **Bit**, then the input can also be a vector of length K .

Upsampling the Modulated Signal

This block can output an upsampled version of the modulated signal. The **Samples per symbol** parameter is the upsampling factor. It must be a positive integer. For more information, see “Upsampled Signals and Rate Changes” on page 2-72.

CPM Modulator Baseband

Dialog Box

Block Parameters: CPM Modulator Baseband

CPM Modulator Baseband (mask)

Output the complex envelope representation of the selected continuous phase modulation.

The input can be either bits or integers. In case of sample-based bit input, the input width must equal the number of bits per symbol. In case of frame-based bit input, the input width must be an integer multiple of the number of bits per symbol. The bits can be either binary-mapped or Gray-mapped into symbols.

For sample-based integer input, the input must be a scalar. For frame-based integer input, the input must be a column vector.

In case of frame-based input, the width of the output frame equals the product of the number of symbols and the Samples per symbol value.

In case of sample-based input, the output sample time equals the symbol period divided by the Samples per symbol value.

The Symbol prehistory parameter is the data symbol(s) used before the start of the simulation.

Parameters

M-ary number:

4

Input type:

Integer

Symbol set ordering:

Binary

Modulation index:

0.5

Frequency pulse shape:

Rectangular

Pulse length (symbol intervals):

1

Symbol prehistory:

1

Phase offset (rad):

0

Samples per symbol:

8

OK

Cancel

Help

Apply

M-ary number

The size of the alphabet.

Input type

Indicates whether the input consists of integers or groups of bits.

Symbol set ordering

Determines how the block maps each group of input bits to a corresponding integer. This field is active only when **Input type** is set to **Bit**.

Modulation index

The number of half-revolutions of phase shift due to the latest symbol when that symbol is the integer 1.

Frequency pulse shape

The type of pulse shaping that the block uses to smooth the phase transitions of the modulated signal.

Rolloff

The rolloff factor of the raised cosine filter. This field appears only when **Frequency pulse shape** is set to **Spectral Raised Cosine**.

BT product

The product of bandwidth and time. This field appears only when **Frequency pulse shape** is set to **Gaussian**.

Pulse length (symbol intervals)

The length of the frequency pulse shape.

Symbol prehistory

The data symbols used before the start of the simulation, in reverse chronological order.

Phase offset (rad)

The initial phase of the output waveform.

Samples per symbol

The number of output samples that the block produces for each integer or binary word in the input.

Pair Block CPM Demodulator Baseband

See Also CPFSK Modulator Baseband, GMSK Modulator Baseband, MSK Modulator Baseband

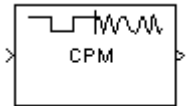
References [1] Anderson, John B., Tor Aulin, and Carl-Erik Sundberg. *Digital Phase Modulation*. New York: Plenum Press, 1986.

CPM Modulator Passband

Purpose Modulate using continuous phase modulation

Library CPM, in Digital Passband sublibrary of Modulation

Description



The CPM Modulator Passband block modulates using continuous phase modulation. The output is a passband representation of the modulated signal. The **M-ary number** parameter, M , is the size of the input alphabet. M must have the form 2^K for some positive integer K .

This block uses the baseband equivalent block, CPM Modulator Baseband, for internal computations and converts the resulting baseband signal to a passband representation. The following parameters in this block are the same as those of the baseband equivalent block:

- **M-ary number**
- **Input type**
- **Symbol set ordering**
- **Modulation index**
- **Frequency pulse shape**
- **Rolloff**
- **BT product**
- **Pulse length**
- **Symbol prehistory**

The input must be sample-based. If the **Input type** parameter is **Bit**, then the input must be a vector of length $\log_2(M)$. If the **Input type** parameter is **Integer**, then the input must be a scalar.

Parameters Specific to Passband Simulation

Passband simulation uses a carrier signal. The **Carrier frequency** and **Carrier initial phase** parameters specify the frequency and initial phase, respectively, of the carrier signal. The **Symbol period** parameter must equal the sample time of the input signal, while the **Output sample time** parameter determines the sample time of the output signal.

This block uses a baseband representation of the modulated signal as an intermediate result during internal computations. The **Baseband samples per symbol** parameter indicates how many baseband samples correspond to each

integer or binary word in the input, before the block converts them to a passband output.

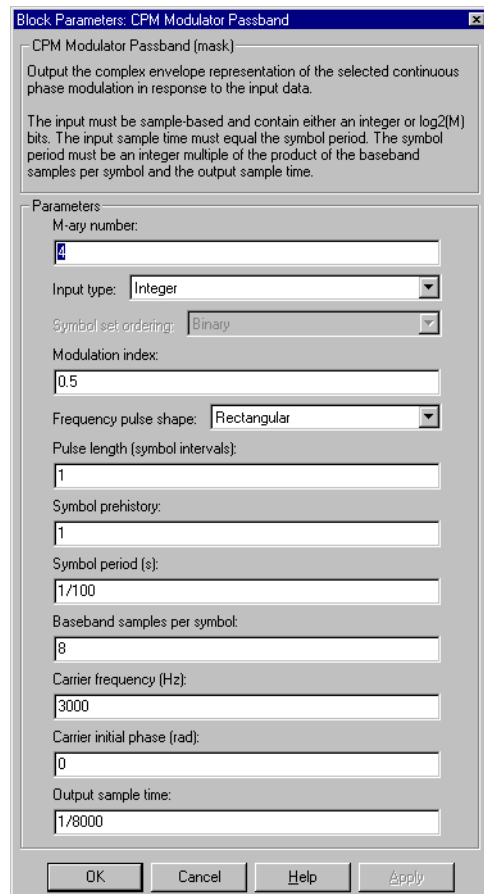
The timing-related parameters must satisfy these relationships:

- **Symbol period** > (**Carrier frequency**)⁻¹
- **Output sample time** < [2***Carrier frequency** + 2/(**Symbol period**)]⁻¹
- **Symbol period** = K***Output sample time*****Baseband samples per symbol**
for some integer K

Furthermore, **Carrier frequency** is typically much larger than the highest frequency of the unmodulated signal.

CPM Modulator Passband

Dialog Box



The dialog box is titled "Block Parameters: CPM Modulator Passband". It contains a description of the block's function and a list of parameters to be configured.

CPM Modulator Passband (mask)
Output the complex envelope representation of the selected continuous phase modulation in response to the input data.
The input must be sample-based and contain either an integer or $\log_2(M)$ bits. The input sample time must equal the symbol period. The symbol period must be an integer multiple of the product of the baseband samples per symbol and the output sample time.

Parameters

- M-ary number: 4
- Input type: Integer
- Symbol set ordering: Binary
- Modulation index: 0.5
- Frequency pulse shape: Rectangular
- Pulse length (symbol intervals): 1
- Symbol prehistory: 1
- Symbol period (s): 1/100
- Baseband samples per symbol: 8
- Carrier frequency (Hz): 3000
- Carrier initial phase (rad): 0
- Output sample time: 1/8000

Buttons: OK, Cancel, Help, Apply

M-ary number

The size of the alphabet.

Input type

Indicates whether the input consists of integers or groups of bits.

Symbol set ordering

Determines how the block maps each group of input bits to a corresponding integer. This field is active only when **Input type** is set to **Bit**.

Modulation index

The number of half-revolutions of phase shift due to the latest symbol when that symbol is the integer 1.

Frequency pulse shape

The type of pulse shaping that the block uses to smooth the phase transitions of the modulated signal.

Rolloff

The rolloff factor of the raised cosine filter. This field appears only when **Frequency pulse shape** is set to **Spectral Raised Cosine**.

BT product

The product of bandwidth and time. This field appears only when **Frequency pulse shape** is set to **Gaussian**.

Pulse length (symbol intervals)

The length of the frequency pulse shape.

Symbol prehistory

The data symbols used before the start of the simulation, in reverse chronological order.

Symbol period (s)

The symbol period, which must equal the sample time of the input.

Baseband samples per symbol

The number of baseband samples that correspond to each integer or binary word in the input, before the block converts them to a passband output.

Carrier frequency (Hz)

The frequency of the carrier.

Carrier initial phase (rad)

The initial phase of the carrier in radians.

Output sample time

The sample time of the output signal.

Pair Block

CPM Demodulator Passband

CPM Modulator Passband

See Also

CPM Modulator Baseband

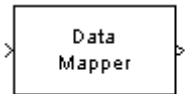
References

[2] Anderson, John B., Tor Aulin, and Carl-Erik Sundberg. *Digital Phase Modulation*. New York: Plenum Press, 1986.

Purpose Map integer symbols from one coding scheme to another

Library Utility Functions

Description The Data Mapper block accepts integer inputs and produces integer outputs. You can select one of four mapping modes: **Binary to Gray**, **Gray to Binary**, **User Defined**, or **Straight Through**.



The input can be either a scalar, a sample-based vector, or a frame-based column vector.

Gray coding is an ordering of binary numbers such that all adjacent numbers differ by only one bit. However, the inputs and outputs of this block are integers, not binary vectors. As a result, the first two mapping modes perform code conversions as follows:

- In the **Binary to Gray** mode, the output from this block is the integer equivalent of the Gray code bit representation for the input integer.
- In the **Gray to Binary** mode, the output from this block is the integer position of the binary equivalent of the input integer in a Gray code ordering.

As an example, the table below shows both the **Binary to Gray** and **Gray to Binary** mappings for integers in the range 0 to 7. In the Binary to Gray Mode Output column, notice that binary representations in successive rows differ by exactly one bit. In the Gray to Binary Mode columns, notice that sorting the rows by Output value creates a Gray code ordering of Input binary representations.

Binary to Gray Mode		Gray to Binary Mode	
Input	Output	Input	Output
0	0 (000)	0 (000)	0
1	1 (001)	1 (001)	1
2	3 (011)	2 (010)	3
3	2 (010)	3 (011)	2
4	6 (110)	4 (100)	7

Data Mapper

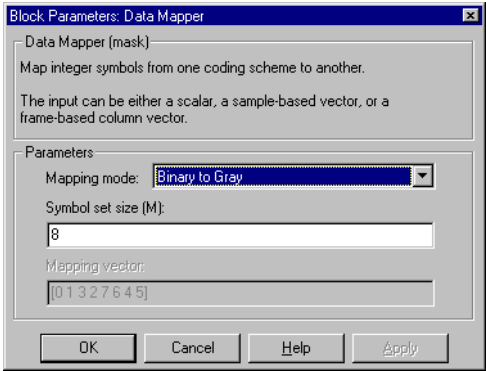
Binary to Gray Mode		Gray to Binary Mode	
Input	Output	Input	Output
5	7 (111)	5 (101)	6
6	5 (101)	6 (110)	4
7	4 (100)	7 (111)	5

When you select the **User Defined** mode, you can use any arbitrary mapping by providing a vector to specify the output ordering. For example, the vector [1, 5, 0, 4, 2, 3] defines the following mapping:

- 0 → 1
- 1 → 5
- 2 → 0
- 3 → 4
- 4 → 2
- 5 → 3

When you select the **Straight Through** mode, the output equals the input.

Dialog Box



Mapping mode

The type of data mapping that the block performs.

Symbol set size

Symbol set size of M restricts this block's inputs and outputs to integers in the range 0 to $M-1$.

Mapping vector

A vector of length M that contains the integers from 0 to $M-1$. The order of the elements of this vector specifies the mapping of inputs to outputs. This field is active only when **Mapping mode** is set to **User Defined**.

DBPSK Demodulator Baseband

Purpose Demodulate DBPSK-modulated data

Library PM, in Digital Baseband sublibrary of Modulation

Description The DBPSK Demodulator Baseband block demodulates a signal that was modulated using the differential binary phase shift keying method. The input is a baseband representation of the modulated signal.



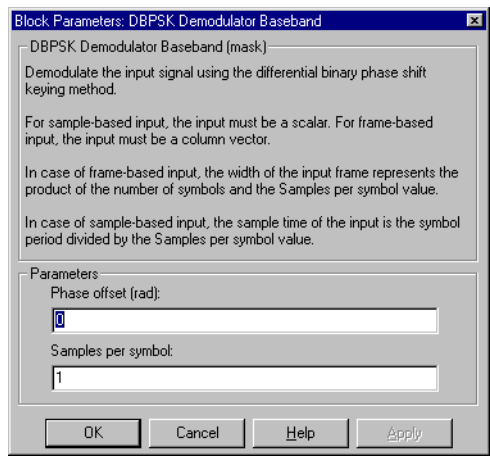
The input must be a discrete-time complex signal. The block compares the current symbol to the previous symbol. It maps phase differences of θ and $\pi+\theta$, respectively, to outputs of 0 and 1, respectively, where θ is the **Phase offset** parameter. The first element of the block’s output is the initial condition of zero because there is no previous symbol with which to compare the first symbol.

The input can be either a scalar or a frame-based column vector.

Processing an Upsampled Modulated Signal

The input signal can be an upsampled version of the modulated signal. The **Samples per symbol** parameter is the upsampling factor. It must be a positive integer. For more information, see “Upsampled Signals and Rate Changes” on page 2-72.

Dialog Box



Phase offset (rad)

This phase difference between the current and previous modulated symbols results in an output of zero.

Samples per symbol

The number of input samples that represent each modulated symbol.

Pair Block DBPSK Modulator Baseband

See Also M-DPSK Demodulator Baseband, DQPSK Demodulator Baseband, BPSK Demodulator Baseband

DBPSK Modulator Baseband

Purpose Modulate using the differential binary phase shift keying method

Library PM, in Digital Baseband sublibrary of Modulation

Description The DBPSK Modulator Baseband block modulates using the differential binary phase shift keying method. The output is a baseband representation of the modulated signal.

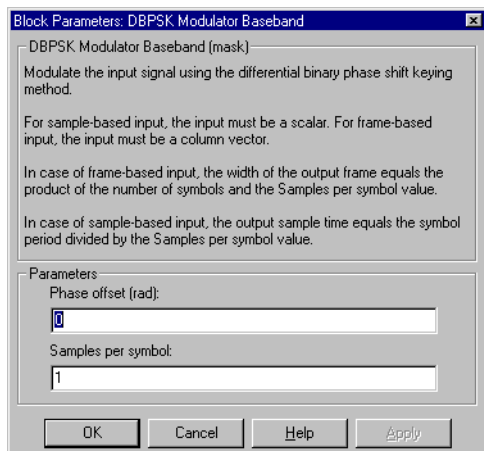


The input must be a discrete-time binary-valued signal. The input can be either a scalar or a frame-based column vector. These rules govern this modulation method when the **Phase offset** parameter is 0:

- If the first input bit is 0 or 1, respectively, then the first modulated symbol is $\exp(j\theta)$ or $-\exp(j\theta)$, respectively.
- If a successive input bit is 0 or 1, respectively, then the modulated symbol is the previous modulated symbol multiplied by $\exp(j\theta)$ or $-\exp(j\theta)$, respectively.

This block can output an upsampled version of the modulated signal. The **Samples per symbol** parameter is the upsampling factor. It must be a positive integer. For more information, see “Upsampled Signals and Rate Changes” on page 2-72.

Dialog Box



Phase offset (rad)

The phase difference between the previous and current modulated symbols when the input is zero.

Samples per symbol

The number of output samples that the block produces for each input bit.

Pair Block

DBPSK Demodulator Baseband

See Also

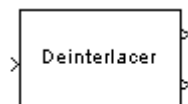
M-DPSK Modulator Passband, DQPSK Modulator Baseband, BPSK Modulator Baseband

Deinterlacer

Purpose Distribute elements of input vector alternately between two output vectors

Library Sequence Operations, in Basic Comm Functions

Description

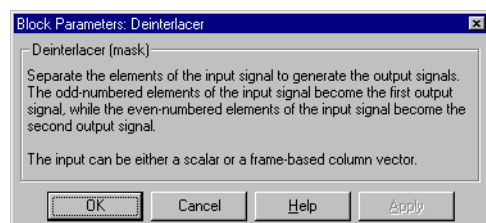


The Deinterlacer block accepts an input vector that has an even number of elements. The block alternately places the elements in each of two output vectors. As a result, each output vector size is half the input vector size. The output vectors have the same complexity and sample time of the input.

The input can be either a sample-based vector of length two, or a frame-based column vector whose length is any even integer.

This block can be useful for separating in-phase and quadrature information from a single vector into separate vectors.

Dialog Box



Examples

If the input vector is frame-based with value [1; 5; 2; 6; 3; 7; 4; 8], then the two output vectors are [1; 2; 3; 4] and [5; 6; 7; 8]. Notice that this is the inverse of the example on the reference page for the Interlacer block.

If the input vector is frame-based with value [1; 2; 3; 4; 5; 6], then the two output vectors are [1; 3; 5] and [2; 4; 6].

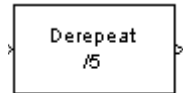
Pair Block Interlacer

See Also Demux (Simulink)

Purpose Reduce sampling rate by averaging consecutive samples

Library Sequence Operations, in Basic Comm Functions

Description



The Derepeat block resamples the discrete input at a rate $1/N$ times the input sample rate by averaging N consecutive samples. This is one possible inverse of the Repeat block (DSP Blockset). The positive integer N is the **Derepeat factor** parameter in the Derepeat mask.

The **Initial condition** parameter prescribes elements of the output when it is still too early for the input data to show up in the output. If the dimensions of the **Initial condition** parameter match the output dimensions, then the parameter represents the initial output value. If **Initial condition** is a scalar, then it represents the initial value of each element in the output.

The input can have any shape or frame status.

Sample-Based Operation

If the input is sample-based, then the block assumes that the input is a vector or matrix whose elements represent samples from independent channels. The block averages samples from each channel independently over time. The output period is N times the input period, and the input and output sizes are identical.

Frame-Based Operation

If the input is frame-based, then the block derepeats each frame, treating distinct channels independently. Each element of the output is the average of N consecutive elements along a *column* of the input matrix. The **Derepeat factor** must be less than the frame size.

The **Framing** parameter determines how the block adjusts the rate at the output to accommodate the reduced number of samples. The two options are:

- **Maintain input frame size**

The block reduces the sampling rate by using a proportionally longer frame *period* at the output port than at the input port. For derepetition by a factor

Derepeat

of N , the output frame period is N times the input frame period, but the input and output frame sizes are equal.

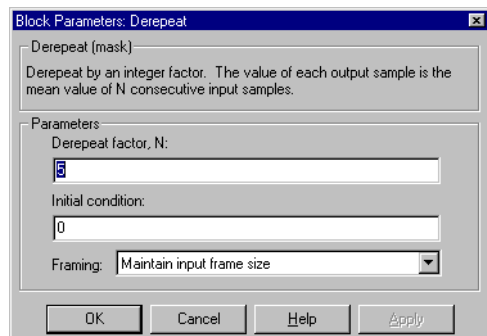
For example, if a single-channel input with a frame period of 1 second is derepeated by a factor of 4, then the output has a frame period of 4 seconds. The input and output frame sizes are equal.

- **Maintain input frame rate**

The block reduces the sampling rate by using a proportionally smaller frame *size* than the input. For derepetition by a factor of N , the output frame size is $1/N$ times the input frame size, but the input and output frame rates are equal. The **Initial condition** parameter does not apply to this option because the input data immediately shows up in the output.

For example, if a single-channel input with 64 elements is derepeated by a factor of 4, then the output contains 16 elements. The input and output frame periods are equal.

Dialog Box



Derepeat factor, N

The number of consecutive input samples to average in order to produce each output sample.

Initial condition

The value with which to initialize the block.

Framing

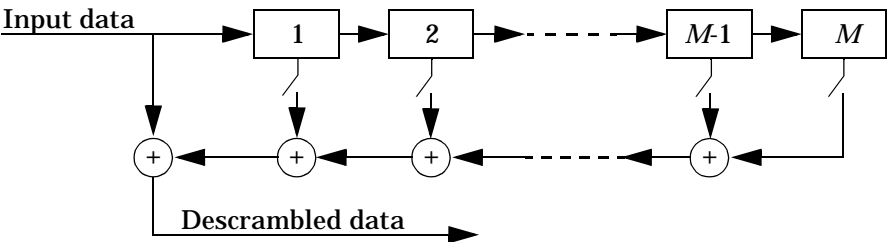
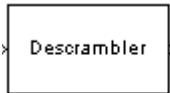
For frame-based operation, the method by which to reduce the amount of data. One method decreases the frame rate while maintaining frame size, while the other decreases the frame size while maintaining frame rate.

See Also

Repeat (DSP Blockset), Downsample (DSP Blockset)

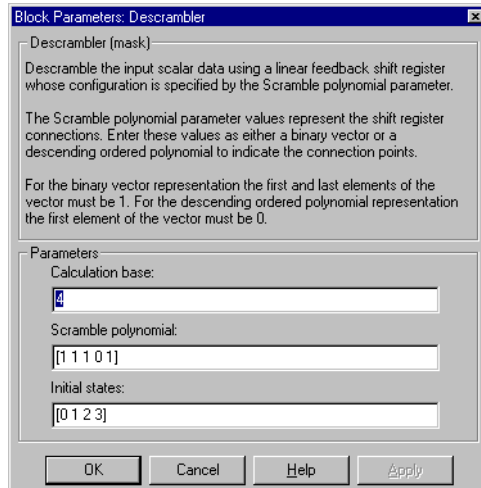
Descrambler

Purpose	Descramble the input signal
Library	Sequence Operations, in Basic Comm Functions
Description	<p>The Descrambler block descrambles the scalar input signal. The Descrambler block is the inverse of the Scrambler block. If you use the Scrambler block in the transmitter, then you should use the Descrambler block in the receiver.</p> <p>Below is a schematic of the descrambler. All adders perform addition modulo N, where N is the Calculation base parameter. The input values must be integers between 0 and N-1.</p>



At each time step, the input causes the contents of the registers to shift sequentially. Each switch in the descrambler is on or off as defined by the **Scramble polynomial** parameter. To make the Descrambler block reverse the operation of the Scrambler block, use the same **Scramble polynomial** parameters in both blocks. The **Initial states** can be different in the two blocks, considering the transmitting and receiving filter delay. See the reference page for the Scrambler block for more information about these parameters.

Dialog Box



Calculation base

The calculation base N . The input and output of this block are integers in the range $[0, N-1]$.

Scramble polynomial

A polynomial that defines the connections in the scrambler.

Initial states

The states of the scrambler's registers when the simulation starts.

Pair Block

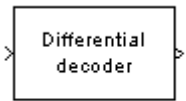
Scrambler

Differential Decoder

Purpose Decode a binary signal using differential coding technique.

Library Source Coding

Description The Differential Decoder block decodes the binary input signal. The output of the Differential Decoder block is the decoded binary signal.



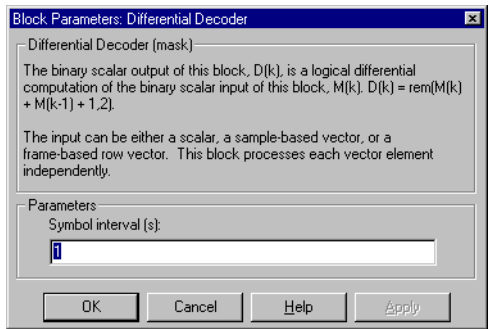
The block's input m and output d are related by

$$d(t_0) = m(t_0) + 1 \mod 2$$
$$d(t_k) = m(t_{k-1}) + m(t_k) + 1 \mod 2$$

where t_k is the k th time step.

The input can be either a scalar, a sample-based vector, or a frame-based row vector. This block processes each vector element independently.

Dialog Box



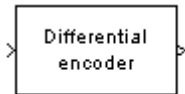
Symbol interval (s)
The sample time of the input symbol.

Pair Block Differential Encoder

Purpose Encode a binary signal using differential coding technique.

Library Source Coding

Description The Differential Encoder block encodes and outputs the binary input signal.



The input m and output d are related by

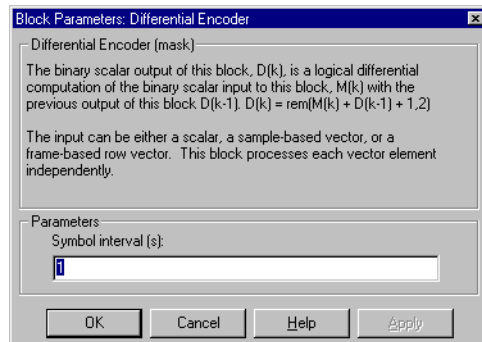
$$d(t_0) = m(t_0) + 1 \mod 2$$

$$d(t_k) = d(t_{k-1}) + m(t_k) + 1 \mod 2$$

where t_k is the k th time step.

The input can be either a scalar, a sample-based vector, or a frame-based row vector. This block processes each vector element independently.

Dialog Box



Symbol interval (s)

The sample time of the input symbol.

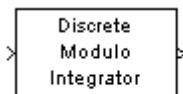
Pair Block Differential Decoder

Discrete Modulo Integrator

Purpose Integrate in discrete time and reduce by a modulus

Library Integrators, in Basic Comm Functions

Description

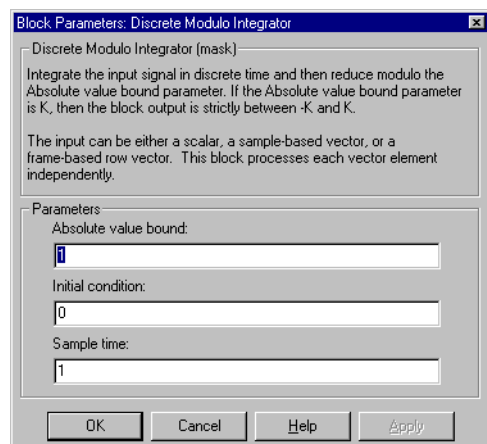


The Discrete Modulo Integrator block integrates its input signal in discrete time and then reduces modulo the **Absolute value bound** parameter. If the **Absolute value bound** parameter is K , then the block output is strictly between $-K$ and K .

The input can be either a scalar, a sample-based vector, or a frame-based row vector. The block processes each vector element independently.

This block's functionality is useful for monotonically increasing or decreasing functions, but works with any integrable function. This block uses the Forward Euler integration method.

Dialog Box



Absolute value bound

The modulus by which the integration result is reduced. This parameter must be nonzero.

Initial condition

The initial condition for integration.

Sample time

The integration sample time.

See Also

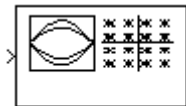
Modulo Integrator, Windowed Integrator, Integrate and Dump, Discrete-Time Integrator (Simulink); rem (MATLAB)

Discrete-Time Eye and Scatter Diagrams

Purpose Produce an eye diagram and/or scatter diagram

Library Comm Sinks

Description



The Discrete-Time Eye and Scatter Diagrams block plots eye diagrams and scatter diagrams from a discrete-time complex input signal. The input can be either a scalar or a frame-based column vector.

The **Diagram type** parameter determines which plots the block produces. The block draws the diagrams in a single window.

The **Trace period** parameter is the number of seconds represented by the horizontal axis in the eye diagram. The **Trace offset** parameter is the time value at the left edge of the horizontal axis of the eye diagram. This parameter must be between zero and the **Trace period** parameter.

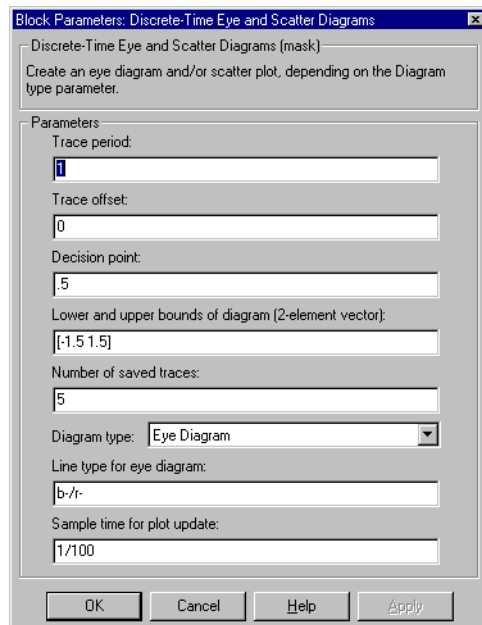
Whenever the simulation time modulo the **Trace period** value equals the **Decision point** parameter, the block plots a new point in the scatter diagram and a vertical line in the eye diagram. The **Decision point** parameter must be greater than or equal to the **Trace offset** parameter, and less than or equal to the sum of **Trace period** and **Trace offset**. Furthermore, if the block plots a scatter diagram, then the **Decision point** parameter must be an integer multiple of the **Sample time for plot update** parameter.

The two-element **Lower and upper bound of incoming signal** vector parameter determines the vertical axis in the eye diagram and both axes in the scatter plot.

To specify the plotting color, as well as the line type or marker type, use the **Line type** parameters that appear after you select a value for the **Diagram type** parameter. In the **Line type for eye diagram** parameter, use a slash (/) to separate the specifications for the in-phase and quadrature components of the input signal. Choices for the color, marker, and line types are in the table on the reference page for the Continuous-Time Eye and Scatter Diagrams block.

The **Sample time for plot update** parameter determines which of the input data the block uses for creating plots. If the parameter matches the sample time of the input signal, then the block uses all available data. If the parameter is an integer multiple of the sample time of the input signal, then the block uses a decimated version of the input data.

Dialog Box



Trace period

The duration of the horizontal axis of the eye diagram, in seconds.

Trace offset

The time at the leftmost edge of the horizontal axis of the eye diagram.

Decision point

The time at which the first point in the scatter plot and the vertical line in the eye diagram are plotted.

Lower and upper bounds of diagram

A two-element vector containing the minimum and maximum signal values in the diagrams.

Number of saved traces

The number of curves in the eye diagram, or points in the scatter plot, that are visible after you resize or restore the figure window.

Discrete-Time Eye and Scatter Diagrams

Diagram type

The diagram(s) that the block produces.

Line type for eye diagram

A string that specifies the color and the line type for the eye diagram. This field appears only when the **Diagram type** parameter is set to an option that includes an eye diagram.

Line type for scatter diagram

A string that specifies the color and the marker type for the scatter diagram. This field appears only when the **Diagram type** parameter is set to an option that includes an scatter diagram.

Sample time for plot update

The time interval between successive input values that the block includes in the plot(s).

Limitations

Since this block uses an M-file, you cannot generate C code for it using the Real-Time Workshop.

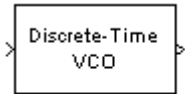
See Also

Continuous-Time Eye and Scatter Diagrams

Purpose Implement a voltage-controlled oscillator in discrete time

Library Comm Sources

Description The Discrete-Time VCO (voltage-controlled oscillator) block generates a signal whose frequency shift from the **Oscillation frequency** parameter is proportional to the input signal. The input signal is interpreted as a voltage. If the input signal is $u(t)$, then the output signal is



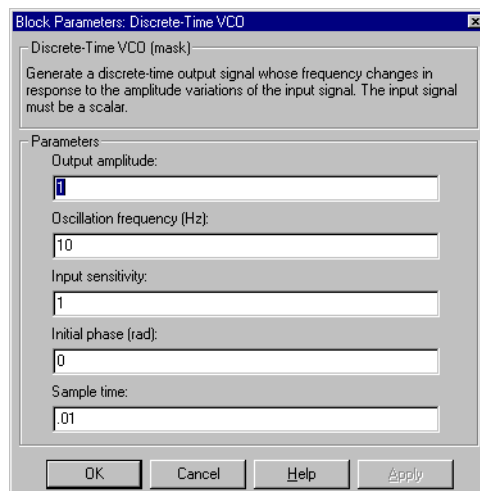
$$y(t) = A_c \cos(2\pi f_c t + 2\pi k_c \int_0^t u(\tau) d\tau + \phi)$$

where A_c is the **Output amplitude**, f_c is the **Oscillation frequency**, k_c is the **Input sensitivity**, and ϕ is the **Initial phase**

This block uses a discrete-time integrator to interpret the equation above.

The input and output signals are both scalars.

Dialog Box



Output amplitude

The amplitude of the output.

Oscillation frequency (Hz)

The frequency of the oscillator output when the input signal is zero.

Discrete-Time VCO

Input sensitivity

This value scales the input voltage and, consequently, the shift from the **Oscillation frequency** value. The units of **Input sensitivity** are Hertz per volt.

Initial phase (rad)

The initial phase of the oscillator in radians.

Sample time

The calculation sample time.

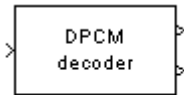
See Also

Voltage-Controlled Oscillator

Purpose Decode differential pulse code modulation

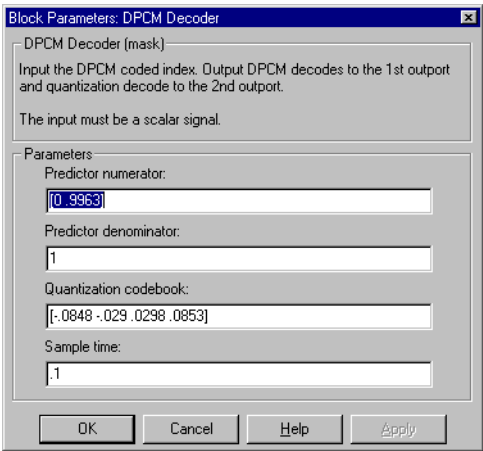
Library Source Coding

Description The DPCM Decoder block recovers a message from a quantized signal using differential pulse code demodulation. The input represents a DPCM-encoded quantization index. The input must be a scalar signal. Its two outputs are the recovered signal and the quantized predictive error.



The description of the Sampled Quantizer Encode block gives more detailed information about quantization indices and quantization-encoded signals. The description of the DPCM Encoder block and the section “Implementing Differential Pulse Code Modulation” on page 2-21 give more information about implementing DPCM.

Dialog Box



Predictor numerator The vector of coefficients of the numerator of the predictor transfer function, in order of ascending powers of z^{-1} . The first entry must be zero.

Predictor denominator The vector of coefficients of the denominator of the predictor transfer function, in order of ascending powers of z^{-1} . Usually this parameter is 1.

DPCM Decoder

Quantization codebook

The vector of output values that the quantizer assigns to each partition.

Sample time

The block's sample time.

Match these parameters to the ones in the corresponding DPCM Encoder block.

Pair Block

DPCM Encoder

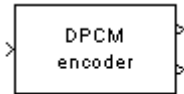
References

[1] Kondo, A. M. *Digital Speech*. Chichester, England: John Wiley & Sons, 1994.

Purpose Encode using differential pulse code modulation

Library Source Coding

Description



The DPCM Encoder block quantizes the input signal using differential pulse code modulation. The input must be a scalar signal. Its two outputs are the quantization index and the quantization-encoded signal.

This block uses the Sampled Quantizer Encode block. The description of that block gives more detailed information about quantization indices and quantization-encoded signals.

Quantization partition is a vector whose entries give the endpoints of the partition intervals. **Quantization codebook**, a vector whose length exceeds the length of **Quantization partition** by one, prescribes a value for each partition in the quantization. The first element of **Quantization codebook** is the value for the interval between negative infinity and the first element of **Quantization partition**.

You can think of the predictor as a transfer function for an IIR filter, hence a rational function of z^{-1} . Specify the predictor's numerator and denominator by listing their coefficients in the vectors **Predictor numerator** and **Predictor denominator**, respectively. List the coefficients in order of increasing powers of z^{-1} .

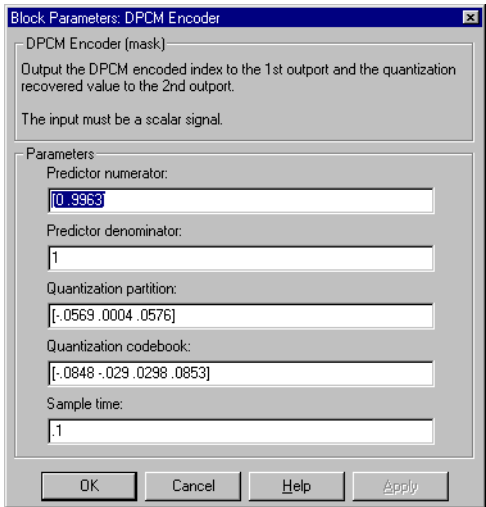
Note The first entry of **Predictor numerator** must be zero. A nonzero entry there would fail to make sense conceptually, and would create an algebraic loop in the implementation.

You can use the function `dpcmopt` in the Communications Toolbox to train the **Predictor numerator**, **Predictor denominator**, **Quantization partition**, and **Quantization codebook** parameters. The output of `dpcmopt` omits the denominator of the predictor, assuming that it will be 1. In most DPCM applications, the denominator of the predictor transfer function is 1.

If **Predictor numerator** has the form $[0, x]$ and **Predictor denominator** is 1, then the modulation is called *delta modulation*.

DPCM Encoder

Dialog Box



Predictor numerator

The vector of coefficients of the numerator of the predictor transfer function, in order of ascending powers of z^{-1} . The first entry must be zero.

Predictor denominator

The vector of coefficients of the denominator of the predictor transfer function, in order of ascending powers of z^{-1} . Usually this parameter is 1.

Quantization partition

The vector of endpoints of the partition intervals. The elements must be in strictly ascending order.

Quantization codebook

The vector of output values that the quantizer assigns to each partition.

Sample time

The block's sample time.

Pair Block

DPCM Decoder

References

[1] Kondo, A. M. *Digital Speech*. Chichester, England: John Wiley & Sons, 1994.

Purpose Demodulate DQPSK-modulated data

Library PM, in Digital Baseband sublibrary of Modulation

Description



The DQPSK Demodulator Baseband block demodulates a signal that was modulated using the differential quaternary phase shift keying method. The input is a baseband representation of the modulated signal.

The input must be a discrete-time complex signal. The output depends on the phase difference between the current symbol and the previous symbol. The first integer (or binary pair, if the **Output type** parameter is set to **Bit**) in the block's output is the initial condition of zero because there is no previous symbol.

The input can be either a scalar or a frame-based column vector.

Outputs and Constellation Types

If the **Output type** parameter is set to **Integer**, then the block maps a phase difference of

$$\theta + \pi m/2$$

to m , where θ is the **Phase offset** parameter and m is 0, 1, 2, or 3.

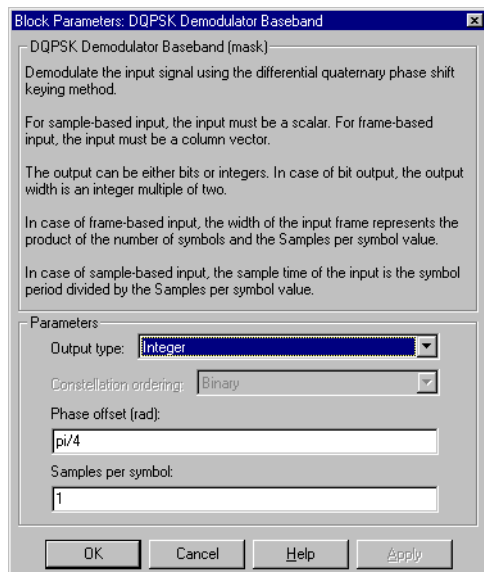
If the **Output type** parameter is set to **Bit**, then the output contains pairs of binary values. The reference page for the DQPSK Modulator Baseband block shows which phase differences map to each binary pair, for the cases when the **Constellation ordering** parameter is either **Binary** or **Gray**.

Processing an Upsampled Modulated Signal

The input signal can be an upsampled version of the modulated signal. The **Samples per symbol** parameter is the upsampling factor. It must be a positive integer. For more information, see “Upsampled Signals and Rate Changes” on page 2-72.

DQPSK Demodulator Baseband

Dialog Box



Output type

Determines whether the output consists of integers or pairs of bits.

Constellation ordering

Determines how the block maps each integer to a pair of output bits. This field is active only when **Output type** is set to **Bit**.

Phase offset (rad)

This phase difference between the current and previous modulated symbols results in an output of zero.

Samples per symbol

The number of input samples that represent each modulated symbol.

Pair Block

DQPSK Modulator Baseband

See Also

M-DPSK Demodulator Baseband, DBPSK Demodulator Baseband, QPSK Demodulator Baseband

Purpose Modulate using the differential quaternary phase shift keying method

Library PM, in Digital Baseband sublibrary of Modulation

Description The DQPSK Modulator Baseband block modulates using the differential quaternary phase shift keying method. The output is a baseband representation of the modulated signal.



The input must be a discrete-time signal.

Inputs and Constellation Types

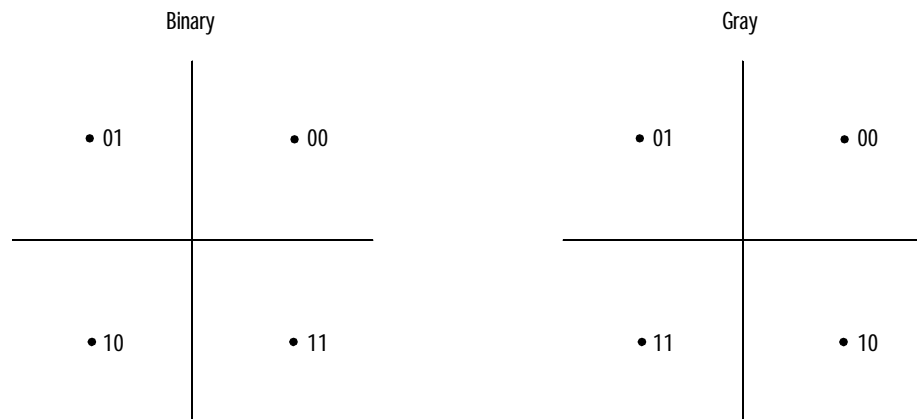
If the **Input type** parameter is set to **Integer**, then valid input values are 0, 1, 2, and 3. In this case, the input can be either a scalar or a frame-based column vector. If the first input is m , then the modulated symbol is

$$\exp(j\theta + j\pi m/2)$$

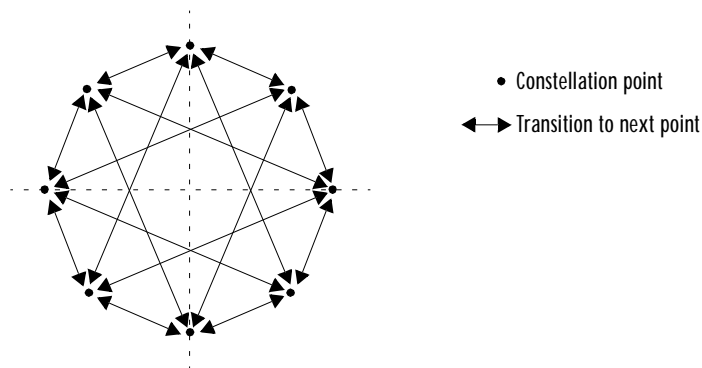
where θ is the **Phase offset** parameter. If a successive input is m , then the modulated symbol is the previous modulated symbol multiplied by $\exp(j\theta + j\pi m/2)$.

If the **Input type** parameter is set to **Bit**, then the input contains pairs of binary values. The input can be either a vector of length two or a frame-based column vector whose length is an even integer. The figure below shows the complex numbers by which the block multiplies the previous symbol to compute the current symbol, depending on whether the **Constellation ordering** parameter is set to **Binary** or **Gray**. The figure assumes that the **Phase offset** parameter is set to $\pi/4$; in other cases, the two schematics would be rotated accordingly.

DQPSK Modulator Baseband



The figure below shows the signal constellation for the DQPSK modulation method when the **Phase offset** parameter is $\pi/4$. The arrows indicate the four possible transitions from each symbol to the next symbol. The **Binary** and **Gray** options determine which transition is associated with each pair of input values.

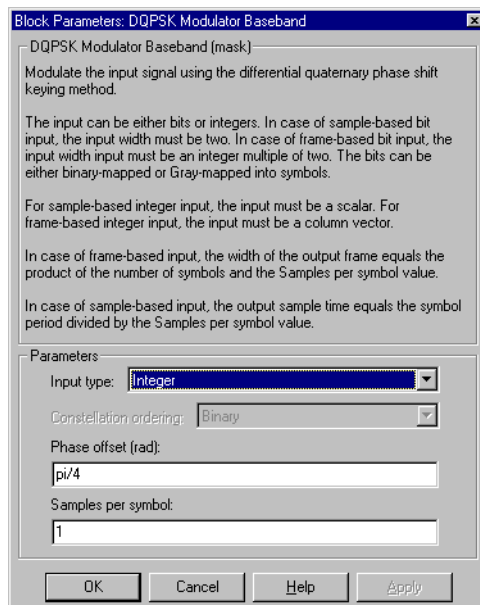


More generally, if the **Phase offset** parameter has the form π/k for some integer k , then the signal constellation has $2k$ points.

Upsampling the Modulated Signal

This block can output an upsampled version of the modulated signal. The **Samples per symbol** parameter is the upsampling factor. It must be a positive integer. For more information, see “Upsampled Signals and Rate Changes” on page 2-72.

Dialog Box



Input type

Indicates whether the input consists of integers or pairs of bits.

Constellation ordering

Determines how the block maps each pair of input bits to a corresponding integer. This field is active only when **Input type** is set to **Bit**.

Phase offset (rad)

The phase difference between the previous and current modulated symbols when the input is zero.

DQPSK Modulator Baseband

Samples per symbol

The number of output samples that the block produces for each integer or pair of bits in the input.

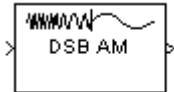
Pair Block DQPSK Demodulator Baseband

See Also M-DPSK Modulator Passband, DBPSK Modulator Baseband, QPSK Modulator Baseband

Purpose Demodulate DSB-AM-modulated data

Library Analog Baseband Modulation, in Modulation

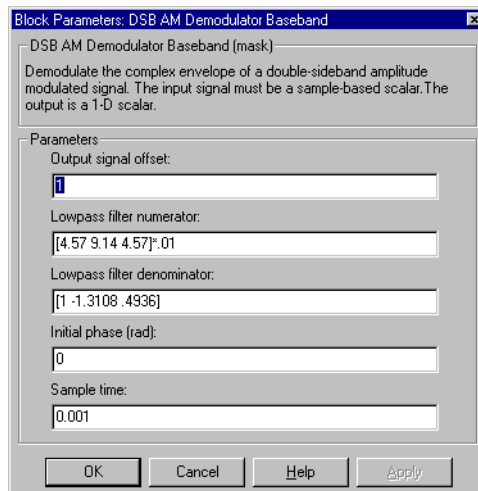
Description



The DSB AM Demodulator Baseband block demodulates a signal that was modulated using double-sideband amplitude modulation. The input is a baseband representation of the modulated signal. The input is complex, while the output is real. The input must be a sample-based scalar signal.

In the course of demodulating, this block uses a filter whose transfer function is described by the **Lowpass filter numerator** and **Lowpass filter denominator** parameters.

Dialog Box



Output signal offset

The same as the **Input signal offset** parameter in the corresponding DSB AM Modulator Baseband block.

Lowpass filter numerator

The numerator of the lowpass filter transfer function. It is represented as a vector that lists the coefficients in order of descending powers of s .

DSB AM Demodulator Baseband

Lowpass filter denominator

The denominator of the lowpass filter transfer function. It is represented as a vector that lists the coefficients in order of descending powers of s . For an FIR filter, set this parameter to 1.

Initial phase (rad)

The initial phase in the corresponding DSB AM Modulator Baseband block.

Sample time

The sample time of the output signal.

Pair Block

DSB AM Modulator Baseband

Purpose	Demodulate DSB-AM-modulated data
Library	Analog Passband Modulation, in Modulation

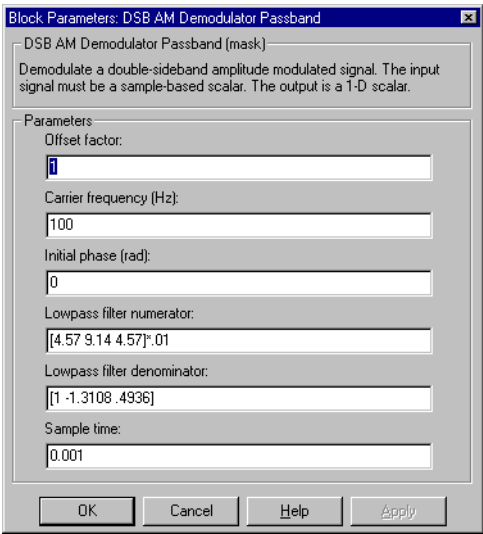
Description



The DSB AM Demodulator Passband block demodulates a signal that was modulated using double-sideband amplitude modulation. The block uses the envelope detection method. The input is a passband representation of the modulated signal. Both the input and output signals are real sample-based scalar signals.

In the course of demodulating, this block uses a filter whose transfer function is described by the **Lowpass filter numerator** and **Lowpass filter denominator** parameters.

Dialog Block



Offset factor

The same as the **Input signal offset** parameter in the corresponding AM with Carrier block.

Carrier frequency (Hz)

The frequency of the carrier in the corresponding AM with Carrier block.

DSB AM Demodulator Passband

Initial phase (rad)

The initial phase of the carrier in radians.

Lowpass filter numerator

The numerator of the lowpass filter transfer function. It is represented as a vector that lists the coefficients in order of descending powers of s .

Lowpass filter denominator

The denominator of the lowpass filter transfer function. It is represented as a vector that lists the coefficients in order of descending powers of s . For an FIR filter, set this parameter to 1.

Sample time

The sample time of the output signal.

Pair Block	DSB AM Modulator Passband
See Also	DSB AM Demodulator Baseband

Purpose Modulate using double-sideband amplitude modulation

Library Analog Baseband Modulation, in Modulation

Description



The DSB AM Modulator Baseband block modulates using double-sideband amplitude modulation. The output is a baseband representation of the modulated signal. The input signal is real, while the output signal is complex. The input must be a sample-based scalar signal.

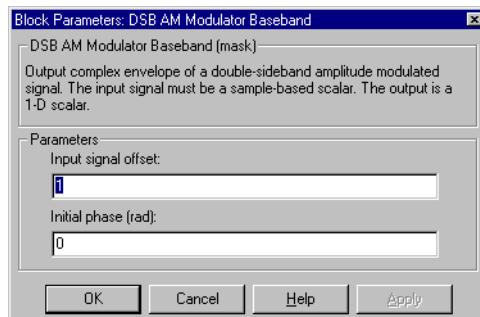
If the input is $u(t)$ as a function of time t , then the output is

$$(u(t) + k)e^{j\theta}$$

where:

- θ is the **Initial phase** parameter.
- k is the **Input signal offset** parameter.

Dialog Box



Input signal offset

The offset factor k . This value should be greater than or equal to the absolute value of the minimum of the input signal.

Initial phase (rad)

The phase of the modulated signal.

Pair Block DSB AM Demodulator Baseband

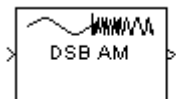
See Also DSBSC AM Modulator Baseband, SSB AM Modulator Baseband

DSB AM Modulator Passband

Purpose Modulate using double-sideband amplitude modulation

Library Analog Passband Modulation, in Modulation

Description



The DSB AM Modulator Passband block modulates using double-sideband amplitude modulation. The output is a passband representation of the modulated signal. Both the input and output signals are real sample-based scalar signals.

If the input is $u(t)$ as a function of time t , then the output is

$$(u(t)+k) \cos(2\pi f_c t + \theta)$$

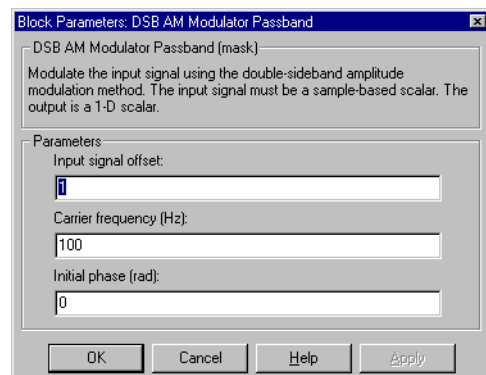
where:

- k is the **Input signal offset** parameter.
- f_c is the **Carrier frequency** parameter.
- θ is the **Initial phase** parameter.

It is common to set the value of k to the maximum absolute value of the negative part of the input signal $u(t)$.

Typically, an appropriate **Carrier frequency** value is much higher than the highest frequency of the input signal. To avoid having to use a high carrier frequency and consequently a high sampling rate, you can use baseband simulation instead of passband simulation.

Dialog Box



Input signal offset

The offset factor k . This value should be greater than or equal to the absolute value of the minimum of the input signal.

Carrier frequency (Hz)

The frequency of the carrier.

Initial phase (rad)

The initial phase of the carrier.

Pair Block DSB AM Demodulator Passband

See Also DSB AM Modulator Baseband, DSBSC AM Modulator Passband, SSB AM Modulator Passband

DSBSC AM Demodulator Baseband

Purpose Demodulate DSBSC-AM-modulated data

Library Analog Baseband Modulation, in Modulation

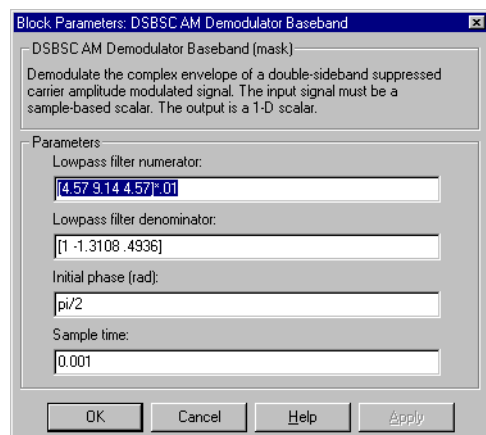
Description



The DSBSC AM Demodulator Baseband block demodulates a signal that was modulated using double-sideband suppressed-carrier amplitude modulation. The input is a baseband representation of the modulated signal. The input is complex, while the output is real. The input must be a sample-based scalar signal.

In the course of demodulating, this block uses a filter whose transfer function is described by the **Lowpass filter numerator** and **Lowpass filter denominator** parameters.

Dialog Box



Lowpass filter numerator

The numerator of the lowpass filter transfer function. It is represented as a vector that lists the coefficients in order of descending powers of s .

Lowpass filter denominator

The denominator of the lowpass filter transfer function. It is represented as a vector that lists the coefficients in order of descending powers of s . For an FIR filter, set this parameter to 1.

Initial phase (rad)

The initial phase in the corresponding DSBSC AM Modulator Baseband block.

Sample time

The sample time of the output signal.

Pair Block DSBSC AM Modulator Baseband

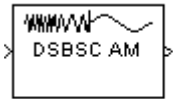
See Also DSB AM Demodulator Baseband, SSB AM Demodulator Baseband

DSBSC AM Demodulator Passband

Purpose Demodulate DSBSC-AM-modulated data

Library Analog Passband Modulation, in Modulation

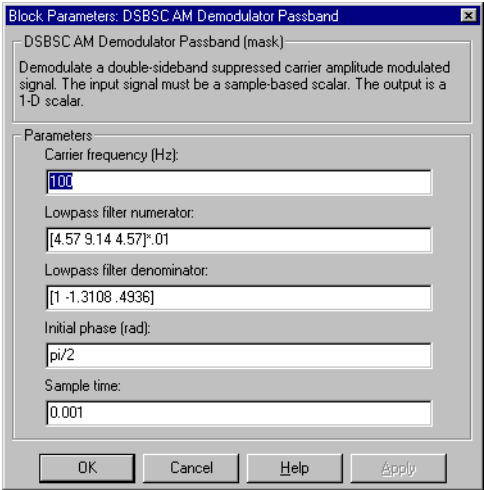
Description



The DSBSC AM Demodulator Passband block demodulates a signal that was modulated using double-sideband suppressed-carrier amplitude modulation. The input is a passband representation of the modulated signal. Both the input and output signals are real sample-based scalar signals.

In the course of demodulating, this block uses a filter whose transfer function is described by the **Lowpass filter numerator** and **Lowpass filter denominator** parameters.

Dialog Box



Carrier frequency (Hz)
The carrier frequency in the corresponding DSBSC AM Modulator Passband block.

Lowpass filter numerator
The numerator of the lowpass filter transfer function. It is represented as a vector that lists the coefficients in order of descending powers of s .

Lowpass filter denominator

The denominator of the lowpass filter transfer function. It is represented as a vector that lists the coefficients in order of descending powers of s . For an FIR filter, set this parameter to 1.

Initial phase (rad)

The initial phase of the carrier in radians.

Sample time

The sample time of the output signal.

Pair Block DSBSC AM Modulator Passband

See Also DSBSC AM Demodulator Baseband, DSB AM Demodulator Passband, SSB AM Demodulator Passband

DSBSC AM Modulator Baseband

Purpose Modulate using double-sideband suppressed-carrier amplitude modulation

Library Analog Baseband Modulation, in Modulation

Description The DSBSC AM Modulator Baseband block modulates using double-sideband suppressed-carrier amplitude modulation. The output is a baseband representation of the modulated signal. The block accepts a real input signal and produces a complex output signal. The input may be continuous-time or discrete-time; the output sample time matches the input sample time. The input must be a sample-based scalar signal.

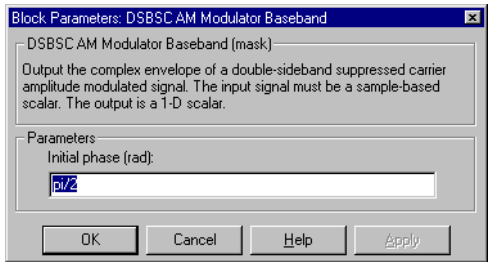


If the input is $u(t)$ as a function of time t , then the output is

$$u(t)e^{j\theta}$$

where θ is the **Initial phase** parameter.

Dialog Box



Initial phase (rad)
The phase of the modulated signal in radians.

Pair Block DSBSC AM Demodulator Baseband

See Also DSB AM Modulator Baseband, SSB AM Modulator Baseband

Purpose Modulate using double-sideband suppressed-carrier amplitude modulation

Library Analog Passband Modulation, in Modulation

Description The DSBSC AM Modulator Passband block modulates using double-sideband suppressed-carrier amplitude modulation. The output is a passband representation of the modulated signal. Both the input and output signals are real sample-based scalar signals.



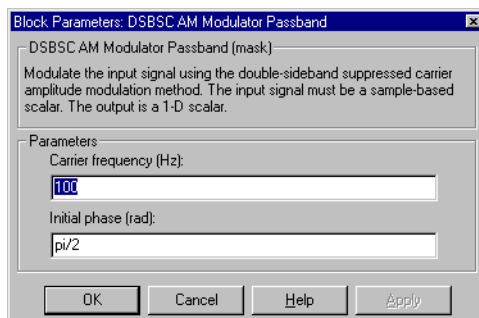
If the input is $u(t)$ as a function of time t , then the output is

$$u(t) \cos(2\pi f_c t + \theta)$$

where f_c is the **Carrier frequency** parameter and θ is the **Initial phase** parameter.

Typically, an appropriate **Carrier frequency** value is much higher than the highest frequency of the input signal. To avoid having to use a high carrier frequency and consequently a high sampling rate, you can use baseband simulation (DSBSC AM Modulator Baseband block) instead of passband simulation.

Dialog Box



Carrier frequency (Hz)

The frequency of the carrier.

Initial phase (rad)

The initial phase of the carrier in radians.

Pair Block DSBSC AM Demodulator Passband

DSBSC AM Modulator Passband

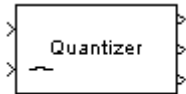
See Also

DSBSC AM Modulator Baseband, DSB AM Modulator Passband, SSB AM Modulator Passband

Purpose Quantize a signal, using trigger to control processing

Library Source Coding

Description

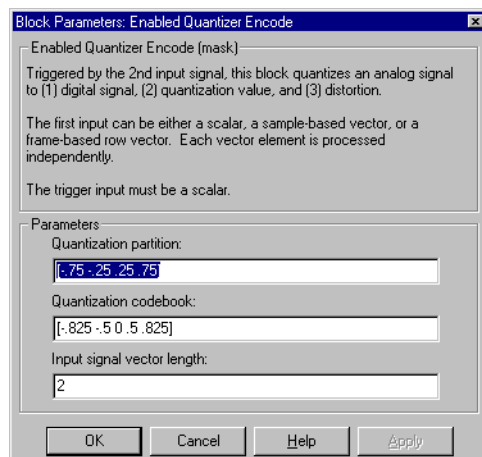


The Enabled Quantizer Encode block performs quantization when a trigger signal occurs. This block is similar to the Sampled Quantizer Encode block, except that a trigger signal at the second input port controls the quantization processing. This block renews its output when the scalar trigger signal is nonzero. For more about quantization, see the reference page for the Sampled Quantizer Encode block.

This block has two input ports and three output ports. The first input signal is the data to be quantized, while the second is the trigger signal that controls the timing of quantization. The three output signals represent the quantization index, quantization value, and mean square distortion, respectively.

The first input can be either a scalar, a sample-based vector, or a frame-based row vector. This block processes each vector element independently. Each output signal is a vector of the same length as the first input signal. The trigger input must be a scalar.

Dialog Box



Enabled Quantizer Encode

Quantization partition

The vector of endpoints of the partition intervals. The elements must be in strictly ascending order.

Quantization codebook

The vector of output values assigned to each partition.

Input signal vector length

The length of the input signal.

Pair Block

Quantizer Decode

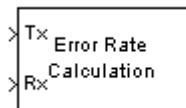
See Also

Sampled Quantizer Encode

Purpose Compute the bit error rate or symbol error rate of input data

Library Comm Sinks

Description



The Error Rate Calculation block compares input data from a transmitter with input data from a receiver. It calculates the error rate as a running statistic, by dividing the total number of unequal pairs of data elements by the total number of input data elements from one source.

You can use this block to compute either symbol or bit error rate, because it does not consider the magnitude of the difference between input data elements. If the inputs are bits, then the block computes the bit error rate. If the inputs are symbols, then it computes the symbol error rate.

This block inherits the sample time of its inputs.

Input Data

This block has between two and four input ports, depending on how you set the mask parameters. The inports marked Tx and Rx accept transmitted and received signals, respectively. The Tx and Rx signals must share the same sampling rate.

The Tx and Rx inputs can be either scalars or frame-based column vectors. If Tx is a scalar and Rx is a vector, or vice-versa, then the block compares the scalar with each element of the vector. (Overall, the block behaves as if you had preprocessed the scalar signal with the DSP Blockset's Repeat block using the **Maintain input frame rate** option.)

If you check the **Reset port** box in the mask, then an additional inport appears, labeled Rst. The Rst input must be a sample-based scalar signal and must have the same sampling rate as the Tx and Rx signals. When the Rst input is nonzero, the block clears its error statistics and then computes them anew.

If you set the **Computation mode** mask parameter to **Select samples from port**, then an additional inport appears, labeled Sel. The Sel input indicates which elements of a frame are relevant for the computation; this is explained further, in the last subbullet below. The Sel input can be either a sample-based column vector or a one-dimensional vector.

Error Rate Calculation

The guidelines below indicate how you should configure the inputs and the mask parameters depending on how you want this block to interpret your Tx and Rx data.

- If both data signals are scalar, then this block compares the Tx scalar signal with the Rx scalar signal. You should leave the **Computation mode** parameter at its default value, **Entire frame**.
- If both data signals are vectors, then this block compares some or all of the Tx and Rx data:
 - If you set the **Computation mode** parameter to **Entire frame**, then the block compares all of the Tx frame with all of the Rx frame.
 - If you set the **Computation mode** parameter to **Select samples from mask**, then the **Selected samples from frame** field appears in the mask. This parameter field accepts a vector that lists the indices of those elements of the Rx frame that you want the block to consider. For example, to consider only the first and last elements of a length-six receiver frame, set the **Selected samples from frame** parameter to [1 6]. If the **Selected samples from frame** vector includes zeros, then the block ignores them.
 - If you set the **Computation mode** parameter to **Select samples from port**, then an additional input port, labeled Sel, appears on the block icon. The data at this input port must have the same format as that of the **Selected samples from frame** mask parameter described above.
- If one data signal is a scalar and the other is a vector, then this block compares the scalar with each entry of the vector. The three subbullets above are still valid for this mode, except that if Rx is a scalar, then the phrase “Rx frame” above refers to the vector expansion of Rx.

Note Simulink requires that input signals have constant length throughout a simulation. If you choose the **Select samples from port** option and want the number of elements in the subframe to vary during the simulation, then you should pad the Sel signal with zeros. (See the Zero Pad block in the DSP Blockset.) The Error Rate Calculation block ignores zeros in the Sel signal.

Output Data

This block produces a vector of length three, whose entries correspond to:

- The error rate
- The total number of errors, that is, comparisons between unequal elements
- The total number of comparisons that the block made

The block sends this output data to the workspace or to an output port, depending on how you set the **Output data** parameter in the mask:

- If you set the **Output data** parameter to **Workspace** and fill in the **Variable name** parameter, then that variable contains the current value when the simulation *ends*. Pausing the simulation does not cause the block to write interim data to the variable.

If you plan to use this block along with the Real-Time Workshop, then you should not use the **Workspace** option; instead, use the **Port** option below and connect the output port to a Simulink To Workspace block.

- If you set the **Output data** parameter to **Port**, then an output port appears. This output port contains the *running* error statistics.

Delays

The **Receive delay** and **Computation delay** parameters implement two different types of delays for this block. One is useful when part of your model causes a lag in the received data, and the other is useful when you want to ignore the transient behavior of both input signals:

- The **Receive delay** parameter is the number of samples by which the received data lags behind the transmitted data. This parameter tells the block which samples “correspond” to each other and should be compared. The receive delay persists throughout the simulation.
- The **Computation delay** parameter tells the block to ignore the specified number of samples at the beginning of the comparison.

Error Rate Calculation

Note The Version 1.4 Error Rate Calculation block considers a vector input to be a sample, whereas the current block considers a vector input to be a frame of multiple samples. For vector inputs of length n , a **Receive delay** of k in the Version 1.4 block is equivalent to a **Receive delay** of $k*n$ in the current block.

If you use the **Select samples from mask** or **Select samples from port** option, then each delay parameter refers to the number of samples that the block receives, whether the block ultimately ignores some of them or not.

Examples

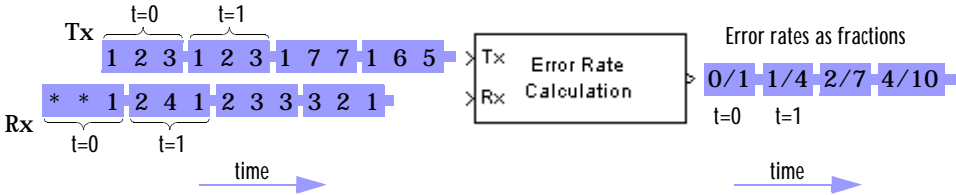
The figure below shows how the block compares pairs of elements and counts the number of error events. This example assumes that the sample time of each input signal is 1 second and that the block's parameters are as follows:

- **Receive delay** = 2
- **Computation delay** = 0
- **Computation mode** = **Entire frame**

The input signals are both frame-based column vectors of length three. However, the schematic arranges each column vector horizontally and aligns pairs of vectors so as to reflect a receive delay of two samples. At each time step, the block compares elements of the Rx signal with those of the Tx signal that appear directly above them in the schematic. For instance, at time 1, the block compares 2, 4, and 1 from the Rx signal with 2, 3, and 1 from the Tx signal.

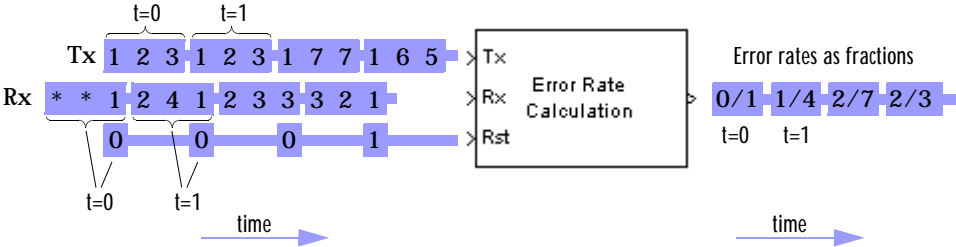
The values of the first two elements of Rx appear as asterisks because they do not influence the output. Similarly, the 6 and 5 in the Tx signal do not influence the output up to time 3, though they *would* influence the output at time 4.

In the error rates on the right side of the figure, each numerator at time t reflects the number of errors when considering the elements of Rx up through time t .



Note: Tx and Rx inputs are frame-based column vectors.

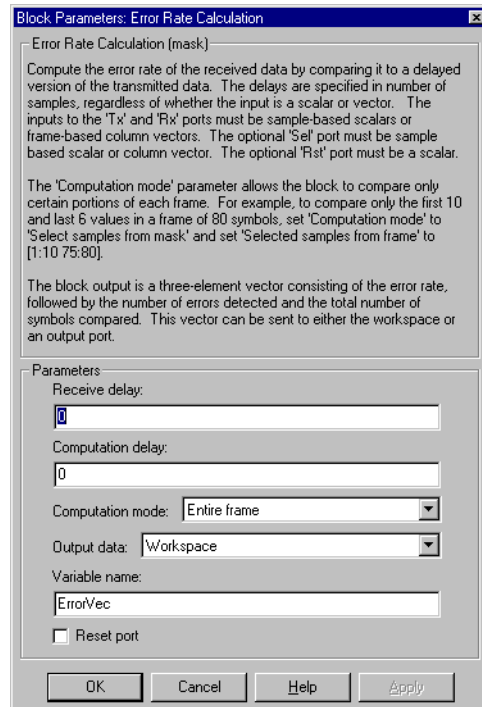
If the block's **Reset port** box had been checked and a reset had occurred at time = 3 seconds, then the last error rate would have been 2/3 instead of 4/10. This value 2/3 would reflect the comparison of 3, 2, and 1 from the Rx signal with 7, 7, and 1 from the Tx signal. The figure below illustrates this scenario.



Note: Tx and Rx inputs are frame-based column vectors.

Error Rate Calculation

Dialog Box



Receive delay

Number of samples by which the received data lags behind the transmitted data. (If Tx or Rx is a vector, then each entry represents a sample.)

Computation delay

Number of samples that the block should ignore at the beginning of the comparison.

Computation mode

Either **Entire frame**, **Select samples from mask**, or **Select samples from port**, depending on whether the block should consider all or only part of the input frames.

Selected samples from frame

A vector that lists the indices of the elements of the Rx frame vector that the block should consider when making comparisons. This field appears only if **Computation mode** is set to **Select samples from mask**.

Output data

Either **Workspace** or **Port**, depending on where you want to send the output data.

Variable name

Name of workspace variable for the output data vector. This field appears only if **Output data** is set to **Workspace**.

Reset port

If you check this box, then an additional input port appears, labeled Rst.

FM Demodulator Baseband

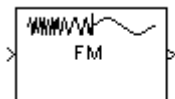
Purpose

Demodulate FM-modulated data

Library

Analog Baseband Modulation, in Modulation

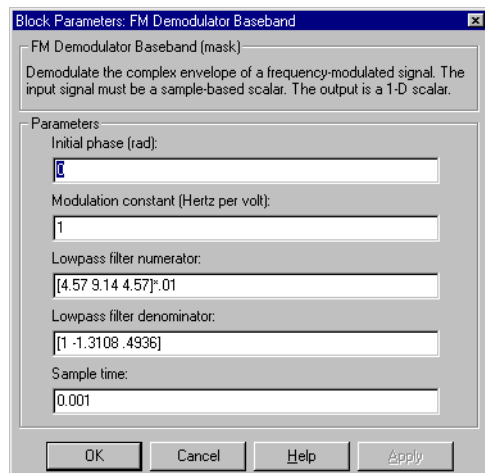
Description



The FM Demodulator Baseband block demodulates a signal that was modulated using frequency modulation. The input is a baseband representation of the modulated signal. The input is complex, while the output is real. The input must be a sample-based scalar signal.

In the course of demodulating, this block uses a filter whose transfer function is described by the **Lowpass filter numerator** and **Lowpass filter denominator** parameters.

Dialog Box



Initial phase (rad)

The initial phase in the corresponding FM Modulator Baseband block.

Modulation constant (Hertz per volt)

The modulation constant in the corresponding FM Modulator Baseband block.

Lowpass filter numerator

The numerator of the lowpass filter transfer function. It is represented as a vector that lists the coefficients in order of descending powers of s .

Lowpass filter denominator

The denominator of the lowpass filter transfer function. It is represented as a vector that lists the coefficients in order of descending powers of s . For an FIR filter, set this parameter to 1.

Sample time

The sample time of the output signal.

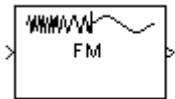
Pair Block FM Modulator Baseband

FM Demodulator Passband

Purpose Demodulate FM-modulated data

Library Analog Passband Modulation, in Modulation

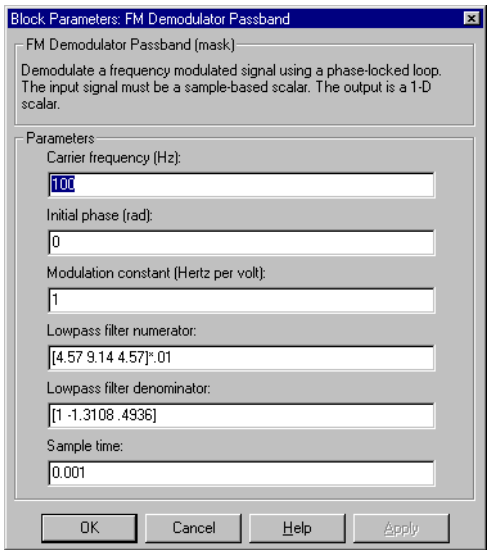
Description The FM Demodulator Passband block demodulates a signal that was modulated using frequency modulation. The input is a passband representation of the modulated signal. Both the input and output signals are real sample-based scalar signals.



In the course of demodulating, the block uses a filter whose transfer function is described by the **Lowpass filter numerator** and **Lowpass filter denominator** parameters.

The block uses a voltage-controlled oscillator (VCO) in the demodulation. The **Initial phase** parameter gives the initial phase of the VCO.

Dialog Box



Carrier frequency (Hz) The carrier frequency in the corresponding FM Modulator Passband block.

Initial phase (rad) The initial phase of the VCO in radians.

Modulation constant (Hertz per volt)

The modulation constant in the corresponding FM Modulator Passband block.

Lowpass filter numerator

The numerator of the lowpass filter transfer function. It is represented as a vector that lists the coefficients in order of descending powers of s .

Lowpass filter denominator

The denominator of the lowpass filter transfer function. It is represented as a vector that lists the coefficients in order of descending powers of s . For an FIR filter, set this parameter to 1.

Sample time

The sample time in the corresponding FM Modulator Passband block.

Pair Block

FM Modulator Passband

See Also

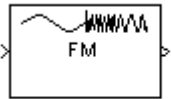
FM Demodulator Baseband

FM Modulator Baseband

Purpose Modulate using frequency modulation

Library Analog Baseband Modulation, in Modulation

Description The FM Modulator Baseband block modulates using frequency modulation. The output is a baseband representation of the modulated signal. The input signal is real, while the output signal is complex. The input must be a sample-based scalar signal. In the frequency modulation technique, the frequency of the modulated signal varies according to the amplitude of the input signal.

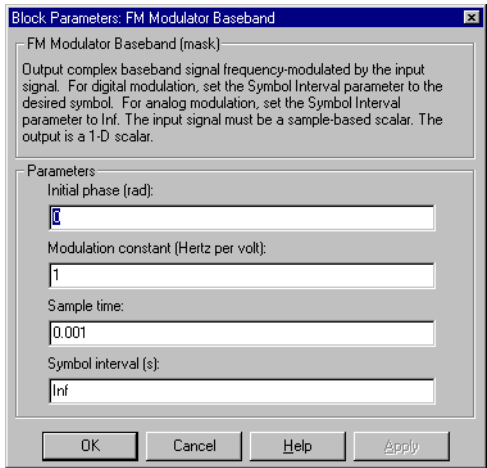


If the input is $u(t)$ as a function of time t , then the output is

$$\exp(j\theta + 2\pi jK_c \int_t u(\tau) d\tau)$$

where θ is the **Initial phase** parameter and K_c is the **Modulation constant** parameter.

Dialog Box



Initial phase (rad)
The initial phase of the modulated signal in radians.

Modulation constant (Hertz per volt)

The modulation constant K_c

Sample time

The sample time of the output signal. It must be a positive number.

Symbol interval (s)

Inf by default. To use this block to model FSK, set this parameter to the length of time required to transmit a single information bit.

Pair Block

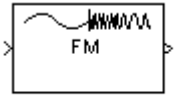
FM Demodulator Baseband

FM Modulator Passband

Purpose Modulate using frequency modulation

Library Analog Passband Modulation, in Modulation

Description



The FM Modulator Passband block modulates using frequency modulation. The output is a passband representation of the modulated signal. The output signal's frequency varies with the input signal's amplitude. Both the input and output signals are real sample-based scalar signals.

If the input is $u(t)$ as a function of time t , then the output is

$$\cos\left(2\pi f_c t + 2\pi K_c \int_t u(\tau) d\tau + \theta\right)$$

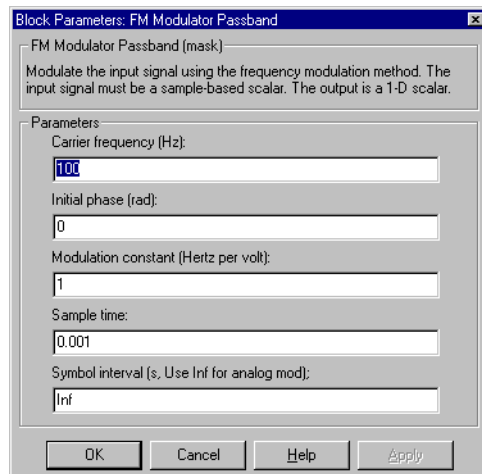
where:

- f_c is the **Carrier frequency** parameter.
- θ is the **Initial phase** parameter.
- K_c is the **Modulation constant** parameter.

Typically, an appropriate **Carrier frequency** value is much higher than the highest frequency of the input signal. To avoid having to use a high carrier frequency and consequently a high sampling rate, you can use baseband simulation (FM Modulator Baseband block) instead of passband simulation.

By the Nyquist sampling theorem, the reciprocal of the **Sample time** parameter must exceed twice the **Carrier frequency** parameter.

Dialog Box



Carrier frequency (Hz)

The frequency of the carrier.

Initial phase (rad)

The initial phase of the carrier in radians.

Modulation constant (Hertz per volt)

The modulation constant K_c .

Sample time

The sample time of the output signal. It must be a positive number.

Symbol interval

Inf by default. To use this block to model FSK, set this parameter to the length of time required to transmit a single information bit.

Pair Block FM Demodulator Passband

See Also FM Modulator Baseband

Gaussian Noise Generator

Purpose Generate Gaussian distributed noise with given mean and variance values

Library Comm Sources

Description



The Gaussian Noise Generator block generates discrete-time white Gaussian noise. You must specify the **Initial seed** vector in the simulation.

The **Mean Value** and the **Variance** can be either scalars or vectors. If either of these is a scalar, then the block applies the same value to each element of a sample-based output or each column of a frame-based output. Individual elements or columns, respectively, are uncorrelated with each other.

When the **Variance** is a vector, its length must be the same as that of the **Initial seed** vector. In this case, the covariance matrix is a diagonal matrix whose diagonal elements come from the **Variance** vector. Since the off-diagonal elements are zero, the output Gaussian random variables are uncorrelated.

When the **Variance** is a square matrix, it represents the covariance matrix. Its off-diagonal elements are the correlations between pairs of output Gaussian random variables. In this case, the **Variance** matrix must be positive definite, and it must be N-by-N, where N is the length of the **Initial seed**.

The probability density function of n -dimensional Gaussian noise is

$$f(x) = \frac{1}{((2\pi)^n \det K)^{\frac{1}{2}}} \exp(-(x-\mu)^T K^{-1}(x-\mu)/2)$$

where x is a length- n vector, K is the n -by- n covariance matrix, μ is the mean value vector, and the superscript T indicates matrix transpose.

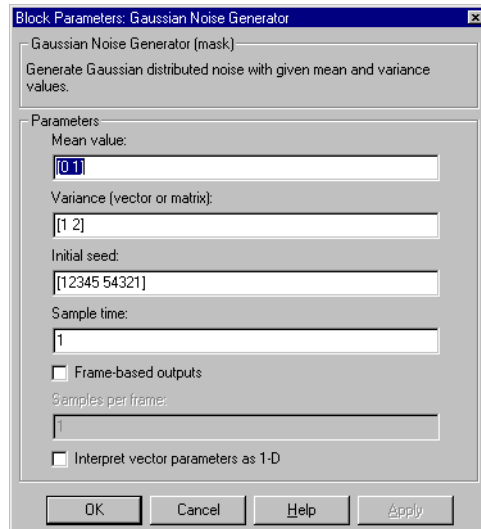
Attributes of Output Signal

The output signal can be a frame-based matrix, a sample-based row or column vector, or a sample-based one-dimensional array. These attributes are controlled by the **Frame-based outputs**, **Samples per frame**, and **Interpret vector parameters as 1-D** parameters. See “Signal Attribute Parameters for Random Sources” on page 2-7 for more details.

If the **Initial seed** parameter is a vector, then its length becomes the number of columns in a frame-based output or the number of elements in a sample-based vector output. In this case, the shape (row or column) of the

Initial seed parameter becomes the shape of a sample-based two-dimensional output signal. If the **Initial seed** parameter is a scalar but either the **Mean value** or **Variance** parameter is a vector, then the vector length determines the output attributes mentioned above.

Dialog Box



Mean value

The mean value of the random variable output.

Variance

The covariance among the output random variables.

Initial seed

The initial seed value for the random number generator.

Sample time

The period of each sample-based vector or each row of a frame-based matrix.

Frame-based outputs

Determines whether the output is frame-based or sample-based. This box is active only if **Interpret vector parameters as 1-D** is unchecked.

Gaussian Noise Generator

Samples per frame

The number of samples in each column of a frame-based output signal. This field is active only if **Frame-based outputs** is checked.

Interpret vector parameters as 1-D

If this box is checked, then the output is a one-dimensional signal. Otherwise, the output is a two-dimensional signal. This box is active only if **Frame-based outputs** is unchecked.

See Also

Random Source (DSP Blockset), AWGN Channel; rand (built-in MATLAB function)

Purpose Restore ordering of the symbols in the input vector

Library Block sublibrary of Interleaving

Description



The General Block Deinterleaver block rearranges the elements of its input vector without repeating or omitting any elements. The input can be real or complex. If the input contains N elements, then the **Elements** parameter is a vector of length N that indicates the indices, in order, of the output elements that came from the input vector. That is, for each integer k between 1 and N ,

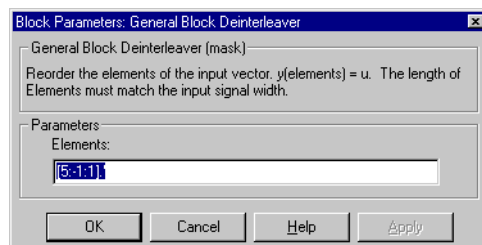
$$\text{Output}(\mathbf{Elements}(k)) = \text{Input}(k)$$

The **Elements** parameter must contain unique integers between 1 and N .

If the input is frame-based, then both it and the **Elements** parameter must be column vectors.

To use this block as an inverse of the General Block Interleaver block, use the same **Elements** parameter in both blocks. In that case, the two blocks are inverses in the sense that applying the General Block Interleaver block followed by the General Block Deinterleaver block leaves data unchanged.

Dialog Box



Elements

A vector of length N that lists the indices of the output elements that came from the input vector.

Examples

This example reverses the operation in the example on the General Block Interleaver block reference page. If **Elements** is $[4, 1, 3, 2]$ and the input to the General Block Deinterleaver block is $[1; 40; 59; 32]$, then the output of the General Block Deinterleaver block is $[40; 32; 59; 1]$.

General Block Deinterleaver

Pair Block General Block Interleaver

See Also perms (MATLAB function)

Purpose Reorder the symbols in the input vector

Library Block sublibrary of Interleaving

Description



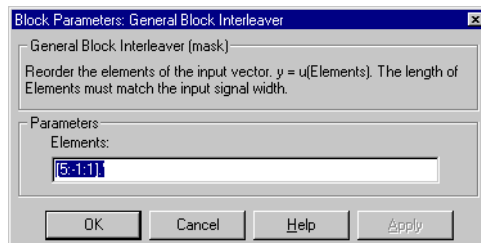
The General Block Interleaver block rearranges the elements of its input vector without repeating or omitting any elements. The input can be real or complex. If the input contains N elements, then the **Elements** parameter is a vector of length N that indicates the indices, in order, of the input elements that form the length- N output vector; that is,

$$\text{Output}(k) = \text{Input}(\mathbf{Elements}(k))$$

for each integer k between 1 and N . The contents of **Elements** must be integers between 1 and N , and must have no repetitions.

If the input is frame-based, then both it and the **Elements** parameter must be column vectors.

Dialog Box



Elements

A vector of length N that lists the indices of the input elements that form the output vector.

Examples

If **Elements** is $[4, 1, 3, 2]$ and the input vector is $[40; 32; 59; 1]$, then the output vector is $[1; 40; 59; 32]$. Notice that all of these vectors have the same length and that the vector **Elements** is a permutation of the vector $[1:4]$.

Pair Block General Block Deinterleaver

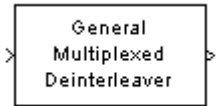
See Also perms (MATLAB function)

General Multiplexed Deinterleaver

Purpose Restore ordering of symbols using specified-delay shift registers

Library Convolutional sublibrary of Interleaving

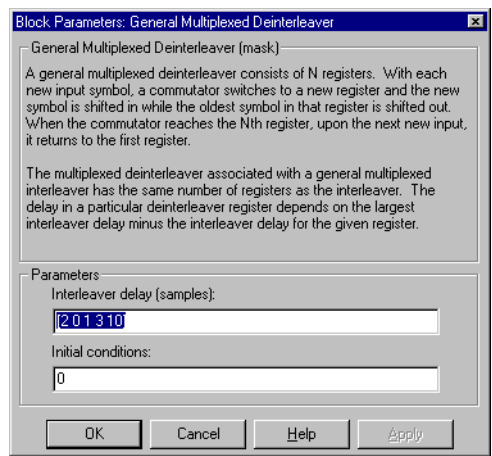
Description The General Multiplexed Deinterleaver block restores the original ordering of a sequence that was interleaved using the General Multiplexed Interleaver block.



In typical usage, the parameters in the two blocks have the same values. As a result, the **Interleaver delay** parameter, V , specifies the delays for each shift register in the corresponding *interleaver*, so that the delays of the deinterleaver's shift registers are actually $\max(V) - V$.

The input can be either a scalar or a frame-based column vector. It can be real or complex. The input and output signals share the same sample time.

Dialog Box



Interleaver delay (samples)

A vector that lists the number of symbols that fit in each shift register of the corresponding interleaver. The length of this vector is the number of shift registers.

Initial conditions

The values that fill each shift register when the simulation begins.

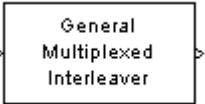
Pair Block General Multiplexed Interleaver

See Also Convolutional Deinterleaver, Helical Deinterleaver

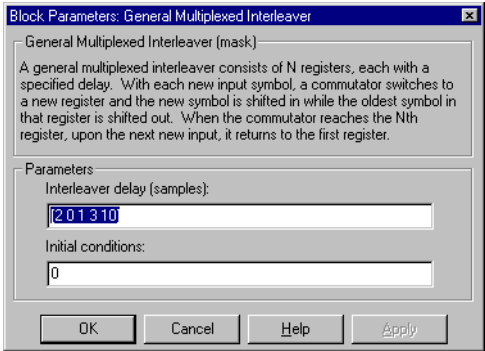
References [1] Heegard, Chris and Stephen B. Wicker. *Turbo Coding*. Boston: Kluwer Academic Publishers, 1999.

General Multiplexed Interleaver

Purpose	Permute input symbols using a set of shift registers with specified delays
Library	Convolutional sublibrary of Interleaving
Description	<p>The General Multiplexed Interleaver block permutes the symbols in the input signal. Internally, it uses a set of shift registers, each with its own delay value.</p> <p>The input can be either a scalar or a frame-based column vector. It can be real or complex. The input and output signals share the same sample time.</p> <p>The Interleaver delay parameter is a column vector whose entries indicate how many symbols can fit into each shift register. The length of the vector is the number of shift registers. (In sample-based mode, it can also be a row vector.)</p> <p>The Initial conditions parameter indicates the values that fill each shift register at the beginning of the simulation. If Initial conditions is a scalar, then its value fills all shift registers; if Initial conditions is a column vector, then each entry fills the corresponding shift register. (In sample-based mode, Initial conditions can also be a row vector.) If a given shift register has zero delay, then the value of the corresponding entry in the Initial conditions vector is unimportant.</p>



Dialog Box



Interleaver delay (samples)

A vector that lists the number of symbols that fit in each shift register. The length of this vector is the number of shift registers.

Initial conditions

The values that fill each shift register when the simulation begins.

Pair Block General Multiplexed Deinterleaver

See Also Convolutional Interleaver, Helical Interleaver

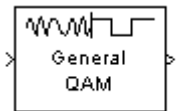
References [1] Heegard, Chris and Stephen B. Wicker. *Turbo Coding*. Boston: Kluwer Academic Publishers, 1999.

General QAM Demodulator Baseband

Purpose Demodulate QAM-modulated data

Library AM, in Digital Baseband sublibrary of Modulation

Description The General QAM Demodulator Baseband block demodulates a signal that was modulated using quadrature amplitude modulation. The input is a baseband representation of the modulated signal.



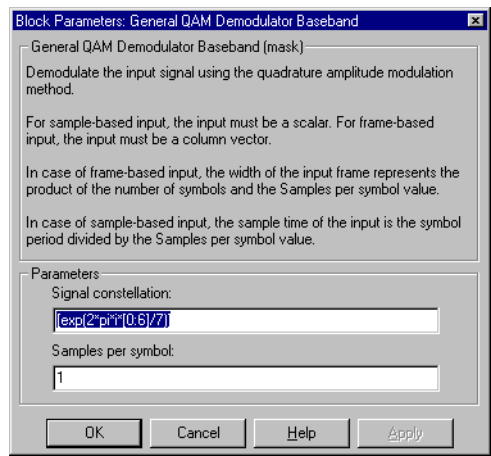
The input must be a discrete-time complex signal. The **Signal constellation** parameter defines the constellation by listing its points in a vector of complex numbers. The block maps the *m*th point in the **Signal constellation** vector to the integer *m*-1.

The input can be either a scalar or a frame-based column vector.

Processing an Upsampled Modulated Signal

The input signal can be an upsampled version of the modulated signal. The **Samples per symbol** parameter is the upsampling factor. It must be a positive integer. For more information, see “Upsampled Signals and Rate Changes” on page 2-72.

Dialog Box



Signal constellation

A real or complex vector that lists the constellation points.

Samples per symbol

The number of input samples that represent each modulated symbol.

Pair Block

General QAM Modulator Baseband

See Also

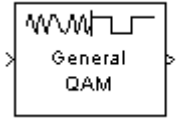
Rectangular QAM Demodulator Baseband

General QAM Demodulator Passband

Purpose Demodulate QAM-modulated data

Library AM, in Digital Passband sublibrary of Modulation

Description



The General QAM Demodulator Passband block demodulates a signal that was modulated using the pulse amplitude phase shift keying method. The input is a passband representation of the modulated signal.

The input must be a sample-based scalar. Furthermore, it must be a discrete-time complex signal.

The **Signal constellation** parameter defines the constellation by listing its points in a vector of complex numbers.

Parameters Specific to Passband Simulation

Passband simulation uses a carrier signal. The **Carrier frequency** and **Carrier initial phase** parameters specify the frequency and initial phase, respectively, of the carrier signal. The **Input sample time** parameter specifies the sample time of the input signal, while the **Symbol period** parameter equals the sample time of the output signal.

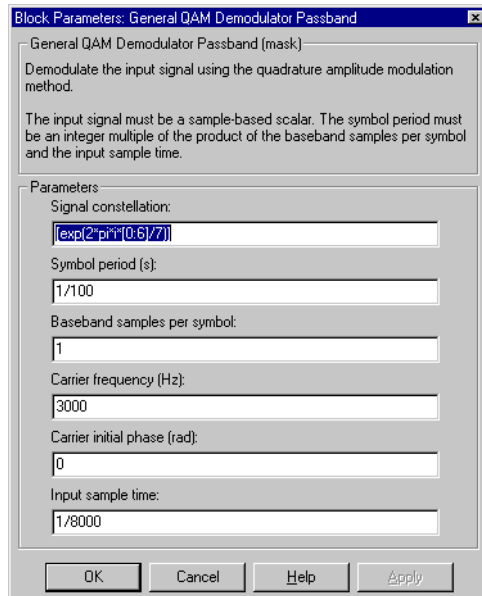
This block uses a baseband representation of the modulated signal as an intermediate signal during internal computations. The **Baseband samples per symbol** parameter indicates how many baseband samples correspond to each integer or binary word in the output.

The timing-related parameters must satisfy these relationships:

- **Input sample time** > (**Carrier frequency**)⁻¹
- **Symbol period** < [2***Carrier frequency** + 2/(**Input sample time**)]⁻¹
- **Input sample time** = K***Symbol period*****Baseband samples per symbol** for some integer K

Also, this block incurs an extra output period of delay compared to its baseband equivalent block.

Dialog Box



Signal constellation

A real or complex vector that lists the constellation points.

Symbol period (s)

The symbol period, which equals the sample time of the output.

Baseband samples per symbol

The number of baseband samples that represent each modulated symbol, after the block converts the passband input to a baseband intermediary signal.

Carrier frequency (Hz)

The frequency of the carrier.

Carrier initial phase (rad)

The initial phase of the carrier in radians.

Input sample time

The sample time of the input signal.

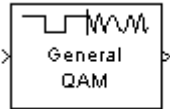
General QAM Demodulator Passband

Pair Block	General QAM Modulator Passband
See Also	General QAM Demodulator Baseband

Purpose Modulate using quadrature amplitude modulation

Library AM, in Digital Baseband sublibrary of Modulation

Description The General QAM Modulator Baseband block modulates using quadrature amplitude modulation. The output is a baseband representation of the modulated signal.



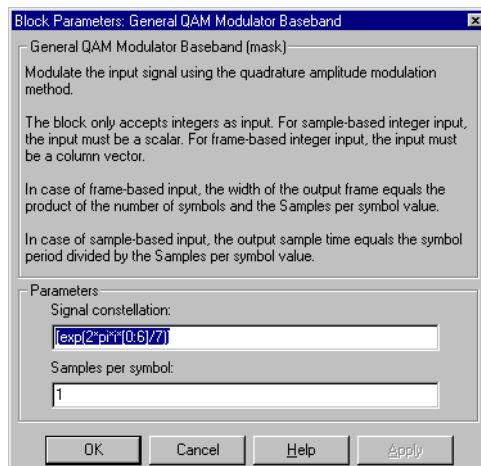
The **Signal constellation** parameter defines the constellation by listing its points in a length-M vector of complex numbers. The input signal values must be integers between 0 and M-1. The block maps an input integer m to the (m-1)st value in the **Signal constellation** vector.

The input can be either a scalar or a frame-based column vector.

Upsampling the Modulated Signal

This block can output an upsampled version of the modulated signal. The **Samples per symbol** parameter is the upsampling factor. It must be a positive integer. For more information, see “Upsampled Signals and Rate Changes” on page 2-72.

Dialog Box



Signal constellation

A real or complex vector that lists the constellation points.

General QAM Modulator Baseband

Samples per symbol

The number of output samples that the block produces for each input integer.

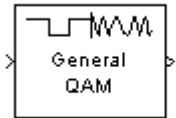
Pair Block General QAM Demodulator Baseband

See Also Rectangular QAM Modulator Baseband

Purpose Modulate using the pulse amplitude modulation phase shift keying method

Library AM, in Digital Passband sublibrary of Modulation

Description



The General QAM Modulator Passband block modulates using the pulse amplitude modulation phase shift keying method. The output is a passband representation of the modulated signal.

The **Signal constellation** parameter defines the constellation by listing its points in a length-M vector of complex numbers. The input signal values must be integers between 0 and M-1. The block maps an input integer m to the (m-1)st value in the **Signal constellation** vector, and then converts these mapped values to a passband output signal.

The input must be a sample-based scalar signal.

Parameters Specific to Passband Simulation

Passband simulation uses a carrier signal. The **Carrier frequency** and **Carrier initial phase** parameters specify the frequency and initial phase, respectively, of the carrier signal. The **Symbol period** parameter must equal the sample time of the input signal, while the **Output sample time** parameter determines the sample time of the output signal.

This block uses a baseband representation of the modulated signal as an intermediate result during internal computations. The **Baseband samples per symbol** parameter indicates how many baseband samples correspond to each integer or binary word in the input, before the block converts them to a passband output.

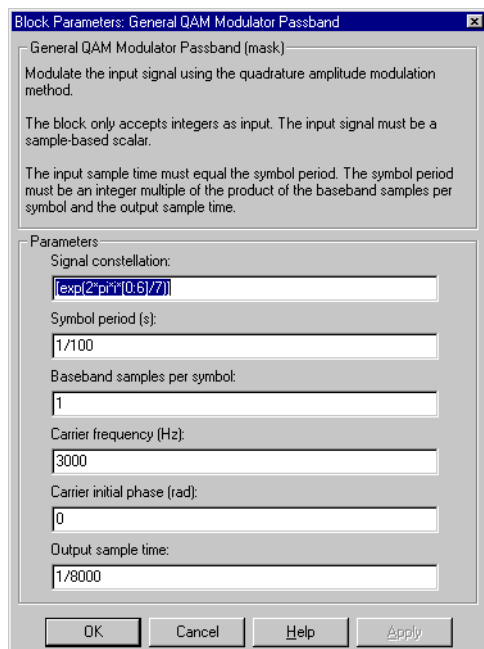
The timing-related parameters must satisfy these relationships:

- **Symbol period** > (**Carrier frequency**)⁻¹
- **Output sample time** < [2***Carrier frequency** + 2/(**Symbol period**)]⁻¹
- **Symbol period** = K***Output sample time*****Baseband samples per symbol**
for some integer K

Furthermore, **Carrier frequency** is typically much larger than the highest frequency of the unmodulated signal.

General QAM Modulator Passband

Dialog Box



Signal constellation

A real or complex vector that lists the constellation points.

Symbol period (s)

The symbol period, which must equal the sample time of the input.

Baseband samples per symbol

The number of baseband samples that correspond to each integer or binary word in the input, before the block converts them to a passband output.

Carrier frequency (Hz)

The frequency of the carrier.

Carrier initial phase (rad)

The initial phase of the carrier in radians.

Output sample time

The sample time of the output signal.

General QAM Modulator Passband

Pair Block General QAM Demodulator Passband

See Also General QAM Modulator Baseband

GMSK Demodulator Baseband

Purpose Demodulate GMSK-modulated data

Library CPM, in Digital Baseband sublibrary of Modulation

Description



The GMSK Demodulator Baseband block demodulates a signal that was modulated using the Gaussian minimum shift keying method. The input is a baseband representation of the modulated signal.

The **BT product**, **Pulse length**, **Symbol prehistory**, and **Phase offset** parameters are as described on the reference page for the GMSK Modulator Baseband block.

Traceback Length and Output Delays

Internally, this block creates a trellis description of the modulation scheme and uses the Viterbi algorithm. The **Traceback length** parameter, D , in this block is the number of trellis branches used to construct each traceback path. D influences the output delay, which is the number of zero symbols that precede the first meaningful demodulated value in the output.

- If the input signal is sample-based, then the delay consists of $D+1$ zero symbols.
- If the input signal is frame-based, then the delay consists of D zero symbols.

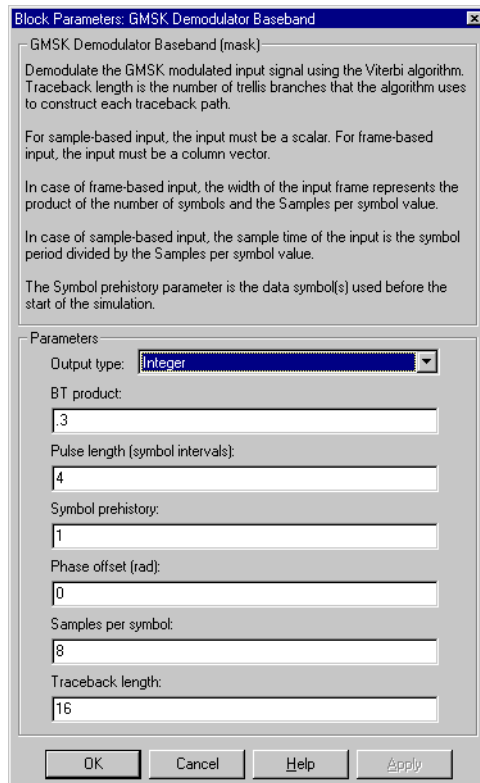
Inputs and Outputs

The input can be either a scalar or a frame-based column vector. If the **Output type** parameter is set to **Integer**, then the block produces values of 1 and -1. If the **Output type** parameter is set to **Bit**, then the block produces values of 0 and 1.

Processing an Upsampled Modulated Signal

The input signal can be an upsampled version of the modulated signal. The **Samples per symbol** parameter is the upsampling factor. It must be a positive integer. For more information, see “Upsampled Signals and Rate Changes” on page 2-72.

Dialog Box



The dialog box is titled "Block Parameters: GMSK Demodulator Baseband". It contains a text area with the following information:

GMSK Demodulator Baseband (mask)

Demodulate the GMSK modulated input signal using the Viterbi algorithm. Traceback length is the number of trellis branches that the algorithm uses to construct each traceback path.

For sample-based input, the input must be a scalar. For frame-based input, the input must be a column vector.

In case of frame-based input, the width of the input frame represents the product of the number of symbols and the Samples per symbol value.

In case of sample-based input, the sample time of the input is the symbol period divided by the Samples per symbol value.

The Symbol prehistory parameter is the data symbol(s) used before the start of the simulation.

Parameters

Output type:

BT product:

Pulse length (symbol intervals):

Symbol prehistory:

Phase offset (rad):

Samples per symbol:

Traceback length:

Buttons: OK, Cancel, Help, Apply

Output type

Determines whether the output consists of bipolar or binary values.

BT product

The product of bandwidth and time.

Pulse length (symbol intervals)

The length of the frequency pulse shape.

Symbol prehistory

The data symbols used by the modulator before the start of the simulation.

Phase offset (rad)

The initial phase of the modulated waveform.

GMSK Demodulator Baseband

Samples per symbol

The number of input samples that represent each modulated symbol.

Traceback length

The number of trellis branches that the Viterbi Decoder block uses to construct each traceback path.

Pair Block GMSK Modulator Baseband

See Also CPM Demodulator Baseband, Viterbi Decoder

References [1] Anderson, John B., Tor Aulin, and Carl-Erik Sundberg. *Digital Phase Modulation*. New York: Plenum Press, 1986.

Purpose Demodulate GMSK-modulated data

Library CPM, in Digital Passband sublibrary of Modulation

Description



The GMSK Demodulator Passband block demodulates a signal that was modulated using the Gaussian minimum shift keying method. The input is a passband representation of the modulated signal.

This block converts the input to an equivalent baseband representation and then uses the baseband equivalent block, GMSK Demodulator Baseband, for internal computations. The following parameters in this block are the same as those of the baseband equivalent block:

- **Output type**
- **BT product**
- **Pulse length**
- **Symbol prehistory**
- **Traceback length**

The input must be a sample-based scalar signal.

Parameters Specific to Passband Simulation

Passband simulation uses a carrier signal. The **Carrier frequency** and **Carrier initial phase** parameters specify the frequency and initial phase, respectively, of the carrier signal. The **Input sample time** parameter specifies the sample time of the input signal, while the **Symbol period** parameter equals the sample time of the output signal.

This block uses a baseband representation of the modulated signal as an intermediate signal during internal computations. The **Baseband samples per symbol** parameter indicates how many baseband samples correspond to each integer or binary word in the output.

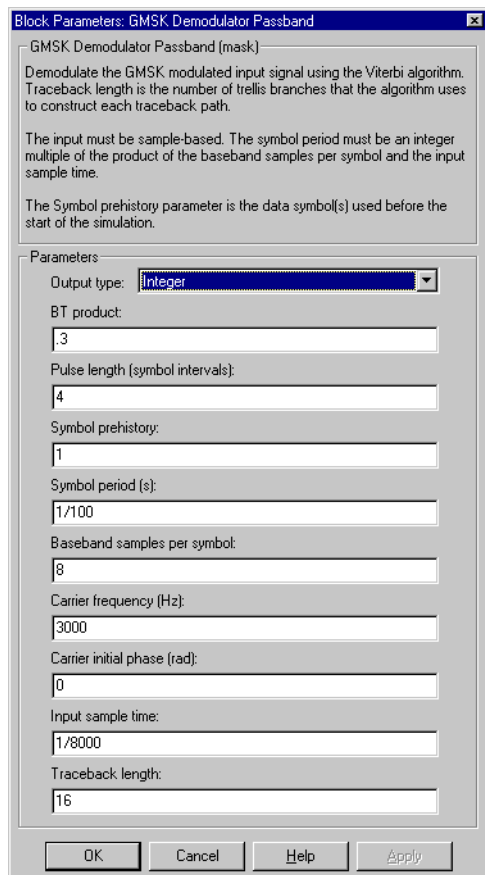
The timing-related parameters must satisfy these relationships:

- **Input sample time** > (**Carrier frequency**)⁻¹
- **Symbol period** < [2***Carrier frequency** + 2/(**Input sample time**)]⁻¹
- **Input sample time** = K***Symbol period*****Baseband samples per symbol** for some integer K

GMSK Demodulator Passband

Also, this block incurs an extra output period of delay compared to its baseband equivalent block.

Dialog Box



The dialog box is titled "Block Parameters: GMSK Demodulator Passband". It contains a description of the block's function and a list of parameters.

GMSK Demodulator Passband (mask)
Demodulate the GMSK modulated input signal using the Viterbi algorithm. Traceback length is the number of trellis branches that the algorithm uses to construct each traceback path.

The input must be sample-based. The symbol period must be an integer multiple of the product of the baseband samples per symbol and the input sample time.

The Symbol prehistory parameter is the data symbol(s) used before the start of the simulation.

Parameters

Output type:	integer
BT product:	.3
Pulse length (symbol intervals):	4
Symbol prehistory:	1
Symbol period (s):	1/100
Baseband samples per symbol:	8
Carrier frequency (Hz):	3000
Carrier initial phase (rad):	0
Input sample time:	1/8000
Traceback length:	16

Buttons: OK, Cancel, Help, Apply

Output type

Determines whether the output consists of bipolar or binary values.

BT product

The product of bandwidth and time.

Pulse length (symbol intervals)

The length of the frequency pulse shape.

Symbol prehistory

The data symbols used by the modulator before the start of the simulation.

Symbol period (s)

The symbol period, which equals the sample time of the output.

Baseband samples per symbol

The number of baseband samples that represent each modulated symbol, after the block converts the passband input to a baseband intermediary signal.

Carrier frequency (Hz)

The frequency of the carrier.

Carrier initial phase (rad)

The initial phase of the carrier in radians.

Input sample time

The sample time of the input signal.

Traceback length

The number of trellis branches that the Viterbi Decoder block uses to construct each traceback path.

Pair Block GMSK Modulator Passband

See Also GMSK Demodulator Baseband, Viterbi Decoder

References [1] Anderson, John B., Tor Aulin, and Carl-Erik Sundberg. *Digital Phase Modulation*. New York: Plenum Press, 1986.

GMSK Modulator Baseband

Purpose Modulate using the Gaussian minimum shift keying method

Library CPM, in Digital Baseband sublibrary of Modulation

Description



The GMSK Modulator Baseband block modulates using the Gaussian minimum shift keying method. The output is a baseband representation of the modulated signal.

The **BT product** parameter represents bandwidth multiplied by time. This parameter is a nonnegative scalar. It is used to reduce the bandwidth at the expense of increased intersymbol interference. The **Pulse length** parameter measures the length of the Gaussian pulse shape, in symbol intervals. For the exact definitions of the pulse shape, see the work by Anderson, Aulin, and Sundberg listed in “References” on page 4-230.

The **Symbol prehistory** parameter is a scalar or vector that specifies the data symbols used before the start of the simulation, in reverse chronological order. If it is a vector, then its length must be one less than the **Pulse length** parameter.

In this block, a symbol of 1 causes a phase shift of $\pi/2$ radians. The **Phase offset** parameter is the initial phase of the output waveform, measured in radians.

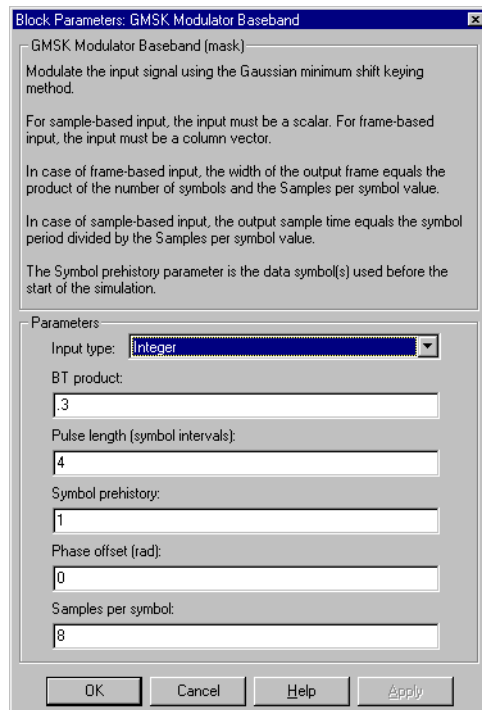
Input Attributes

The input can be either a scalar or a frame-based column vector. If the **Input type** parameter is set to **Integer**, then the block accepts values of 1 and -1. If the **Input type** parameter is set to **Bit**, then the block accepts values of 0 and 1.

Upsampling the Modulated Signal

This block can output an upsampled version of the modulated signal. The **Samples per symbol** parameter is the upsampling factor. It must be a positive integer. For more information, see “Upsampled Signals and Rate Changes” on page 2-72.

Dialog Box



Input type

Indicates whether the input consists of bipolar or binary values.

BT product

The product of bandwidth and time.

Pulse length (symbol intervals)

The length of the frequency pulse shape.

Symbol prehistory

The data symbols used before the start of the simulation, in reverse chronological order.

Phase offset (rad)

The initial phase of the output waveform.

GMSK Modulator Baseband

Samples per symbol

The number of output samples that the block produces for each integer or bit in the input.

Pair Block GMSK Demodulator Baseband

See Also CPM Modulator Baseband

References [1] Anderson, John B., Tor Aulin, and Carl-Erik Sundberg. *Digital Phase Modulation*. New York: Plenum Press, 1986.

Purpose Modulate using the Gaussian minimum shift keying method

Library CPM, in Digital Passband sublibrary of Modulation

Description



The GMSK Modulator Passband block modulates using the Gaussian minimum shift keying method. The output is a passband representation of the modulated signal.

This block uses the baseband equivalent block, GMSK Modulator Baseband, for internal computations and converts the resulting baseband signal to a passband representation. The following parameters in this block are the same as those of the baseband equivalent block:

- **Input type**
- **BT product**
- **Pulse length**
- **Symbol prehistory**

The input must be sample-based. If the **Input type** parameter is **Bit**, then the input must be a vector of length $\log_2(M)$. If the **Input type** parameter is **Integer**, then the input must be a scalar.

Parameters Specific to Passband Simulation

Passband simulation uses a carrier signal. The **Carrier frequency** and **Carrier initial phase** parameters specify the frequency and initial phase, respectively, of the carrier signal. The **Symbol period** parameter must equal the sample time of the input signal, while the **Output sample time** parameter determines the sample time of the output signal.

This block uses a baseband representation of the modulated signal as an intermediate result during internal computations. The **Baseband samples per symbol** parameter indicates how many baseband samples correspond to each integer or binary word in the input, before the block converts them to a passband output.

The timing-related parameters must satisfy these relationships:

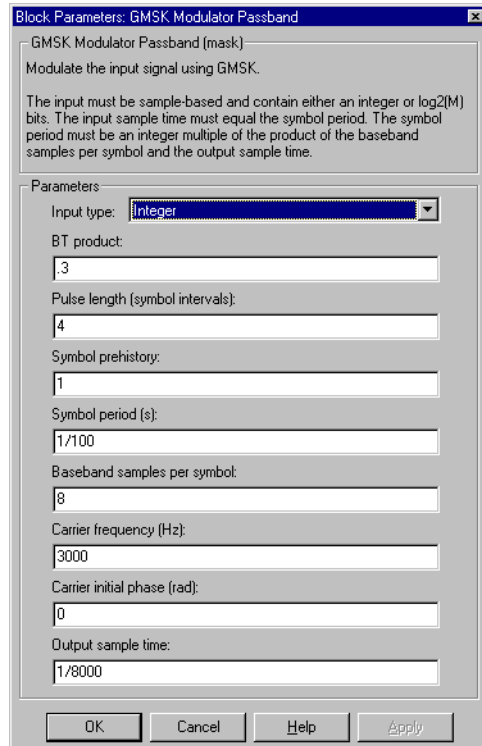
- **Symbol period** > $(\text{Carrier frequency})^{-1}$
- **Output sample time** < $[2 * \text{Carrier frequency} + 2/(\text{Symbol period})]^{-1}$

GMSK Modulator Passband

- **Symbol period** = $K \times \text{Output sample time} \times \text{Baseband samples per symbol}$ for some integer K

Furthermore, **Carrier frequency** is typically much larger than the highest frequency of the unmodulated signal.

Dialog Box



The dialog box is titled "Block Parameters: GMSK Modulator Passband". It contains a description of the block's function and a list of parameters to be configured.

GMSK Modulator Passband (mask)
Modulate the input signal using GMSK.

The input must be sample-based and contain either an integer or log2(M) bits. The input sample time must equal the symbol period. The symbol period must be an integer multiple of the product of the baseband samples per symbol and the output sample time.

Parameters

- Input type:
- BT product:
- Pulse length (symbol intervals):
- Symbol prehistory:
- Symbol period (s):
- Baseband samples per symbol:
- Carrier frequency (Hz):
- Carrier initial phase (rad):
- Output sample time:

Buttons: OK, Cancel, Help, Apply

Input type

Indicates whether the input consists of bipolar or binary values.

BT product

The product of bandwidth and time.

Pulse length (symbol intervals)

The length of the frequency pulse shape.

Symbol prehistory

The data symbols used before the start of the simulation, in reverse chronological order.

Symbol period (s)

The symbol period, which must equal the sample time of the input.

Baseband samples per symbol

The number of baseband samples that correspond to each integer or binary word in the input, before the block converts them to a passband output.

Carrier frequency (Hz)

The frequency of the carrier.

Carrier initial phase (rad)

The initial phase of the carrier in radians.

Output sample time

The sample time of the output signal.

Pair Block GMSK Demodulator Passband

See Also GMSK Modulator Baseband

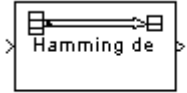
References [1] Anderson, John B., Tor Aulin, and Carl-Erik Sundberg. *Digital Phase Modulation*. New York: Plenum Press, 1986.

Hamming Decoder

Purpose Decode a Hamming code to recover binary vector data

Library Block sublibrary of Channel Coding

Description



The Hamming Decoder block recovers a binary message vector from a binary Hamming codeword vector. For proper decoding, the parameter values in this block should match those in the corresponding Hamming Encoder block.

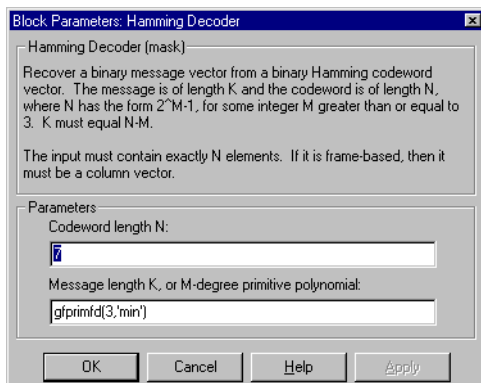
If the Hamming code has message length K and codeword length N , then N must have the form $2^M - 1$ for some integer M greater than or equal to 3. Also, K must equal $N - M$.

The input must contain exactly N elements. If it is frame-based, then it must be a column vector. The output is a vector of length K .

The coding scheme uses elements of the finite field $GF(2^M)$. You can either specify the primitive polynomial that the algorithm should use, or you can rely on the default setting:

- To use the default primitive polynomial, simply enter N and K as the first and second mask parameters, respectively. The algorithm uses `gfprimdf(M)` as the primitive polynomial for $GF(2^M)$.
- To specify the primitive polynomial, enter N as the first parameter and a binary vector as the second parameter. The vector represents the primitive polynomial by listing its coefficients in order of ascending exponents. You can create primitive polynomials using the `gfprimdf` function in the Communications Toolbox.

Dialog Box



Codeword length N

The codeword length N, which is also the input vector length.

Message length K, or M-degree primitive polynomial

Either the message length, which is also the output vector length; or a binary vector that represents a primitive polynomial for $GF(2^M)$.

Pair Block

Hamming Encoder

See Also

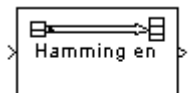
hammgen (Communications Toolbox)

Hamming Encoder

Purpose Create a Hamming code from binary vector data

Library Block sublibrary of Channel Coding

Description



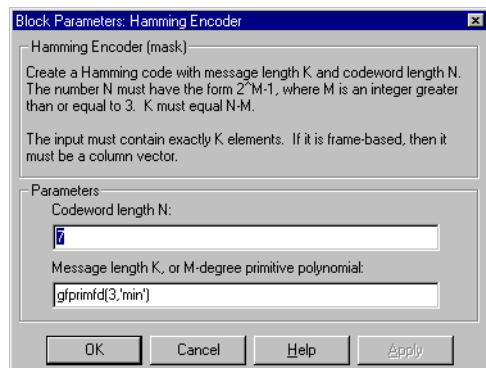
The Hamming Encoder block creates a Hamming code with message length K and codeword length N . The number N must have the form $2^M - 1$, where M is an integer greater than or equal to 3. Then K equals $N - M$.

The input must contain exactly K elements. If it is frame-based, then it must be a column vector. The output is a vector of length N .

The coding scheme uses elements of the finite field $GF(2^M)$. You can either specify the primitive polynomial that the algorithm should use, or you can rely on the default setting:

- To use the default primitive polynomial, simply enter N and K as the first and second mask parameters, respectively. The algorithm uses `gfprimdf(M)` as the primitive polynomial for $GF(2^M)$.
- To specify the primitive polynomial, enter N as the first parameter and a binary vector as the second parameter. The vector represents the primitive polynomial by listing its coefficients in order of ascending exponents. You can create primitive polynomials using the `gfprimdf` function in the Communications Toolbox.

Dialog Box



Codeword length N

The codeword length, which is also the output vector length.

Message length K, or M-degree primitive polynomial

Either the message length, which is also the input vector length; or a binary vector that represents a primitive polynomial for $\text{GF}(2^M)$.

Pair Block Hamming Decoder

See Also `hammgen` (Communications Toolbox)

Helical Deinterleaver

Purpose Restore ordering of symbols permuted by a helical interleaver

Library Convolutional sublibrary of Interleaving

Description



The Helical Deinterleaver block permutes the symbols in the input signal by placing them in an array row by row and then selecting groups in a helical fashion to send to the output port.

The block uses the array internally for its computations. If **C** is the **Number of columns in helical array** parameter, then the array has **C** columns and unlimited rows. If **N** is the **Group size** parameter, then the block accepts an input of length **C*N** at each time step and inserts them into the next **N** rows of the array. The block also places the **Initial condition** parameter into certain positions in the top few rows of the array (not only to accommodate the helical pattern but also to preserve the vector indices of symbols that pass through the Helical Interleaver and Helical Deinterleaver blocks in turn).

The output consists of consecutive groups of **N** symbols. Counting from the beginning of the simulation, the block selects the **k**th output group in the array from column **k mod C**. The selection is helical because of the reduction modulo **C** and because the first symbol in the **k**th group is in row $1+(k-1)*s$, where **s** is the **Helical array step size** parameter.

The number of elements of the input vector must be **C** times **N**. If the input is frame-based, then it must be a column vector.

Delay of Interleaver-Deinterleaver Pair

After processing a message with the Helical Interleaver block and the Helical Deinterleaver block, the deinterleaved data lags the original message by

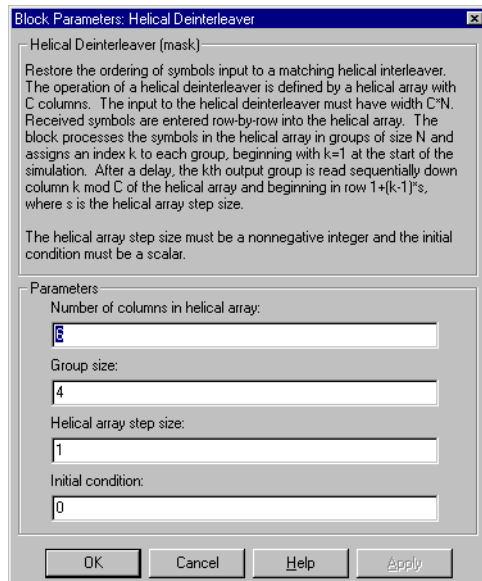
$$CN \left\lceil \frac{s(C-1)}{N} \right\rceil$$

samples. Before this delay elapses, the deinterleaver output is either the **Initial condition** parameter in the Helical Deinterleaver block or the **Initial condition** parameter in the Helical Interleaver block.

If your model incurs an additional delay between the interleaver output and the deinterleaver input, then the restored sequence lags the original sequence by the sum of the additional delay and the amount in the formula above. For proper synchronization, the delay between the interleaver and deinterleaver

must be $m \cdot C \cdot N$ for some nonnegative integer m . You can use the Integer Delay block in the DSP Blockset to adjust delays manually, if necessary.

Dialog Box



Number of columns in helical array

The number of columns, C , in the helical array.

Group size

The size, N , of each group of symbols. The input width is C times N .

Helical array step size

The number of rows of separation between consecutive output groups as the block selects them from their respective columns of the helical array.

Initial condition

A scalar that fills the array before the first input is placed.

Pair Block

Helical Interleaver

See Also

General Multiplexed Deinterleaver

Helical Deinterleaver

References

[1] Berlekamp, E. R. and P. Tong. "Improved Interleavers for Algebraic Block Codes." U. S. Patent 4559625, Dec. 17, 1985.

Purpose Permute input symbols using a helical array

Library Convolutional sublibrary of Interleaving

Description



The Helical Interleaver block permutes the symbols in the input signal by placing them in an array in a helical fashion and then sending rows of the array to the output port.

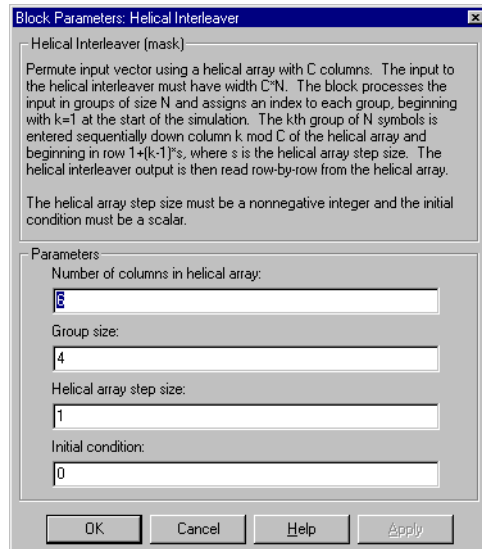
The block uses the array internally for its computations. If C is the **Number of columns in helical array** parameter, then the array has C columns and unlimited rows. If N is the **Group size** parameter, then the block accepts an input of length $C*N$ at each time step and partitions the input into consecutive groups of N symbols. Counting from the beginning of the simulation, the block places the k th group in the array along column $k \bmod C$. The placement is helical because of the reduction modulo C and because the first symbol in the k th group is in row $1+(k-1)*s$, where s is the **Helical array step size** parameter. Positions in the array that do not contain input symbols have default contents specified by the **Initial condition** parameter.

The block sends $C*N$ symbols from the array to the output port by reading the next N rows sequentially. At a given time step, the output symbols might be the **Initial condition** parameter value, symbols from that time step's input vector, or symbols left in the array from a previous time step.

The number of elements of the input vector must be C times N . If the input is frame-based, then it must be a column vector.

Helical Interleaver

Dialog Box



Number of columns in helical array

The number of columns, C , in the helical array.

Group size

The size, N , of each group of input symbols. The input width is C times N .

Helical array step size

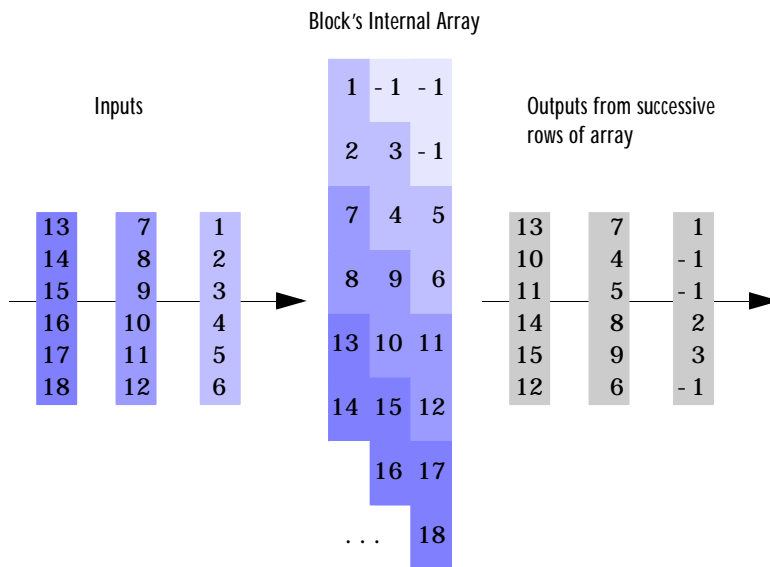
The number of rows of separation between consecutive input groups in their respective columns of the helical array.

Initial condition

A scalar that fills the array before the first input is placed.

Examples

Suppose that $C = 3$, $N = 2$, the **Helical array step size** parameter is 1, and the **Initial condition** parameter is -1. After receiving inputs of $[1: 6]'$, $[7: 12]'$, and $[13: 19]'$, the block's internal array looks like the schematic below. The coloring of the inputs and the array indicate how the input symbols are placed within the array. The outputs at the first three time steps are $[1; -1; -1; 2; 3; -1]$, $[7; 4; 5; 8; 9; 6]$, and $[13; 10; 11; 14; 15; 12]$. (The outputs are not color-coded in the schematic.)



Pair Block

Helical Deinterleaver

See Also

General Multiplexed Interleaver

References

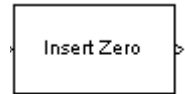
[1] Berlekamp, E. R. and P. Tong, "Improved Interleavers for Algebraic Block Codes." U. S. Patent 4559625, Dec. 17, 1985.

Insert Zero

Purpose Distribute input elements in output vector

Library Sequence Operations, in Basic Comm Functions

Description



The Insert Zero block constructs an output vector by inserting zeros among the elements of the input vector. The input can be real or complex. The block determines where to place the zeros by using the **Insert zero vector** parameter. The **Insert zero vector** parameter is a binary vector whose elements are arranged so that:

- Each 1 indicates that the block should place the *next* element of the input in the output vector
- Each 0 indicates that the block should place a 0 in the output vector

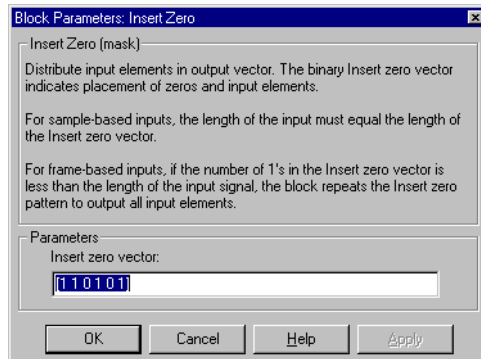
If the input signal is sample-based, then the input vector length must equal the number of 1s in the **Insert zero vector** parameter.

To implement punctured coding using the Puncture and Insert Zero blocks, you should use the same vector for the **Insert zero vector** parameter in this block and for the **Puncture vector** parameter in the Puncture block.

Frame-Based Processing

If the input signal is frame-based, then both it and the **Insert zero vector** parameter must be column vectors. The number of 1s in the **Insert zero vector** parameter must divide the input vector length. If the input vector length is greater than the number of 1s in the **Insert zero vector** parameter, then the block repeats the insertion pattern until it has placed all input elements in the output vector.

Dialog Box



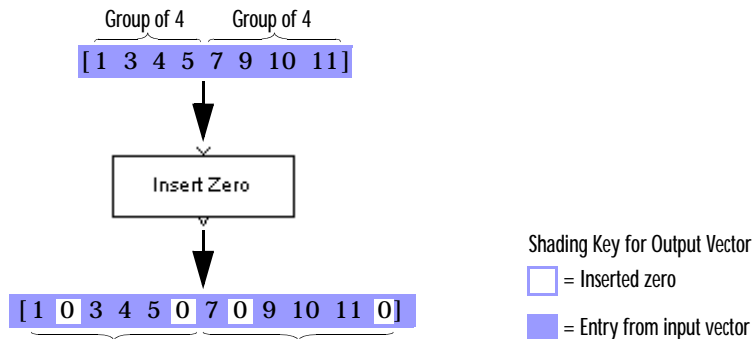
Insert zero vector

A binary vector whose pattern of 0s and 1s indicates where the block should place either 0s or input vector elements, respectively, in the output vector.

Examples

If the **Insert zero vector** parameter is the six-element vector [1, 0, 1, 1, 1, 0], then the block inserts zeros after the first and last elements of each consecutive grouping of four input elements. It considers groups of four elements because the **Insert zero vector** parameter has four 1s.

The diagram below depicts the block's operation using this **Insert zero vector** parameter. Notice that the insertion pattern applies twice.



Compare this example with that on the reference page for the Puncture block.

See Also

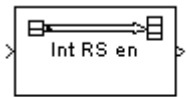
Puncture

Integer-Input RS Encoder

Purpose Create a Reed-Solomon code from integer vector data

Library Block sublibrary of Channel Coding

Description



The Integer-Input RS Encoder block creates a Reed-Solomon code with message length K and codeword length N . You specify both N and K directly in the block mask. N must have the form $2^M - 1$, where M is an integer greater than or equal to 3. The code is more efficient if $N - K$ is an even integer.

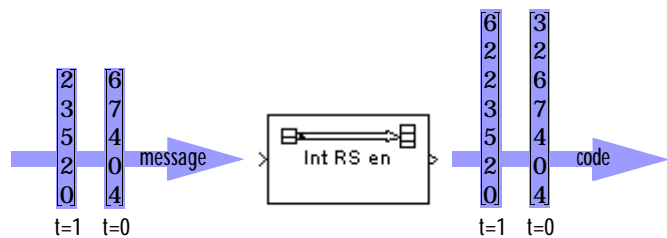
The input and output are integer-valued signals that represent messages and codewords, respectively. The integers that form the messages and codewords represent elements of the finite field $GF(2^M)$ and hence must be between 0 and $2^M - 1$.

The input must contain exactly K elements. If it is frame-based, then it must be a column vector. The output is a vector of length N .

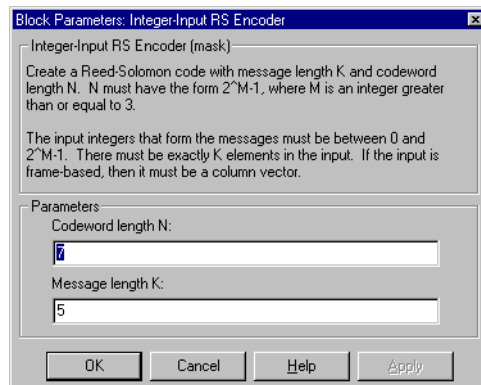
An (N, K) Reed-Solomon code can correct up to $\text{floor}((N - K) / 2)$ symbol errors (*not* bit errors) in each codeword.

Examples

Suppose $M = 3$, $N = 2^3 - 1 = 7$, and $K = 5$. Then a message is a vector of length 5 whose entries are integers between 0 and 7. A corresponding codeword is a vector of length 7 whose entries are integers between 0 and 7. The figure below illustrates possible input and output signals to this block, when the block parameters are set to their defaults of 7 and 5, respectively.



Dialog Box



Codeword length N

The codeword length, which is also the output vector length.

Message length K

The message length, which is also the input vector length.

Pair Block

Integer-Output RS Decoder

See Also

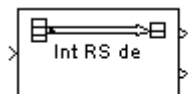
Binary-Input RS Encoder

Integer-Output RS Decoder

Purpose Decode a Reed-Solomon code to recover integer vector data

Library Block sublibrary of Channel Coding

Description



The Integer-Output RS Decoder block recovers a message vector from a Reed-Solomon codeword vector. For proper decoding, the parameter values in this block should match those in the corresponding Integer-Input RS Encoder block.

If the Reed-Solomon code has message length K and codeword length N , then N must have the form $2^M - 1$ for some integer M greater than or equal to 3. The code is more efficient if $N - K$ is an even integer.

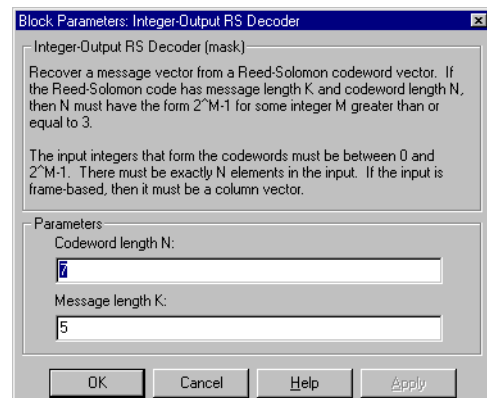
The input and first output are integer-valued signals that represent codewords and messages, respectively. The integers in these signals represent elements of the finite field $GF(2^M)$ and hence must be between 0 and $2^M - 1$.

The input must contain exactly N elements. If it is frame-based, then it must be a column vector. The first output is a vector of length K .

The second output is the number of errors detected during decoding of the codeword. A negative integer indicates that the block detected more errors than it could correct using the coding scheme. An (N, K) Reed-Solomon code can correct up to $\text{floor}((N - K) / 2)$ symbol errors (*not* bit errors) in each codeword.

The sample times of the input and output signals are equal.

Dialog Box



Codeword length N

The codeword length, which is also the input vector length.

Message length K

The message length, which is also the output vector length.

Pair Block Integer-Input RS Encoder

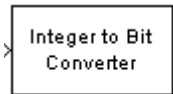
See Also Binary-Output RS Decoder

Integer to Bit Converter

Purpose Map a vector of integers to a vector of bits

Library Utility Functions

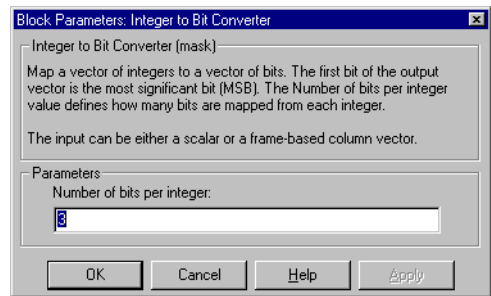
Description



The Integer to Bit Converter block maps each integer in the input vector to a group of bits in the output vector. If M is the **Number of bits per integer** parameter, then the input integers must be between 0 and 2^M-1 . The block maps each integer to a group of M bits, using the first bit as the most significant bit. As a result, the output vector length is M times the input vector length.

The input can be either a scalar or a frame-based column vector.

Dialog Box



Number of bits per integer

The number of bits the block uses to represent each integer of the input. This parameter must be an integer between 1 and 31.

Examples

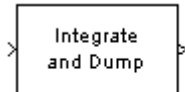
If the input is $[7; 13]$ and the **Number of bits per integer** parameter is 4, then the output is $[0; 1; 1; 1; 1; 1; 0; 1]$. The first group of four bits (0, 1, 1, 1) represents 7 and the second group of four bits (1, 1, 0, 1) represents 13. Notice that the output length is four times the input length.

Pair Block Bit to Integer Converter

Purpose Integrate, resetting to zero periodically and reducing by a modulus

Library Integrators, in Basic Comm Functions

Description



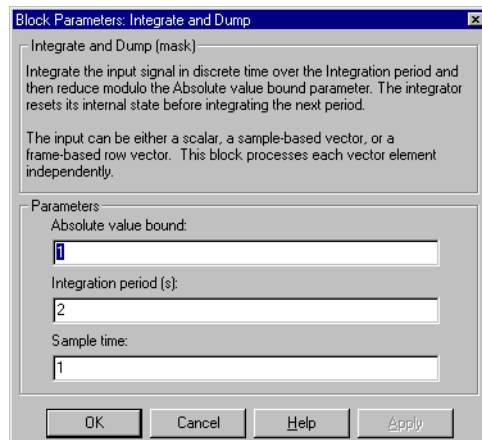
The Integrate and Dump block integrates the input signal in discrete time, resets to zero according to a fixed schedule, and reduces modulo the **Absolute value bound** parameter. If the **Absolute value bound** parameter is K , then the block output is strictly between $-K$ and K .

The reset times are the positive integral multiples of the **Integration period** parameter. At each reset time, the block performs its final integration step, sends the result to the output port, and then clears its internal state for the next time step.

The input can be either a scalar, a sample-based vector, or a frame-based row vector. The block processes each vector element independently.

This block uses the Forward Euler integration method.

Dialog Box



Absolute value bound

The modulus by which the integration result is reduced. This parameter must be positive.

Integrate and Dump

Integration period (s)

The first reset time. This is also the time interval between resets.

Sample time

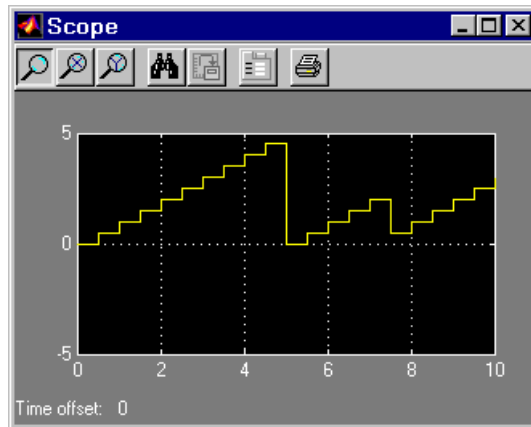
The integration sample time. This must not exceed the **Integration period**.

Examples

Integrate a constant signal whose value is 1 using these parameters:

- **Absolute value bound** = 5
- **Integration period** = 7
- **Sample time** = . 5

You can use a Simulink Constant block for the input signal. The Simulink Scope block shows the output below.



Notice that the output is 0 at time 0 and that the output never exceeds 5. Also notice that the output at time 7.5 seconds (**Integration period** plus **Sample time**) is the result of resetting the integrator after the previous time step and then considering the input signal between times 7 and 7.5.

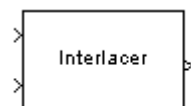
See Also

Discrete Modulo Integrator, Windowed Integrator, Discrete-Time Integrator (Simulink)

Purpose Alternately select elements from two input vectors to generate output vector

Library Sequence Operations, in Basic Comm Functions

Description

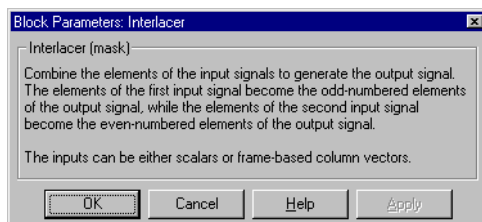


The Interlacer block accepts two inputs that have the same vector size, complexity, and sample time. It produces one output vector by alternating elements from the first input and from the second input. As a result, the output vector size is twice that of either input. The output vector has the same complexity and sample time of the inputs.

The inputs can be either scalars or frame-based column vectors.

This block can be useful for combining in-phase and quadrature information from separate vectors into a single vector.

Dialog Box



Examples If the two input vectors are frame-based with values [1; 2; 3; 4] and [5; 6; 7; 8], then the output vector is [1; 5; 2; 6; 3; 7; 4; 8].

Pair Block Deinterlacer

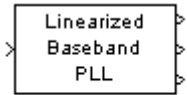
See Also General Block Interleaver; Mux (Simulink)

Linearized Baseband PLL

Purpose Implement a linearized version of a baseband phase-locked loop

Library Synchronization

Description



The Linearized Baseband PLL block is a feedback control system that automatically adjusts the phase of a locally generated signal to match the phase of an input signal. Unlike the Phase-Locked Loop block, this block uses a baseband model method. Unlike the Baseband PLL block, which uses a nonlinear model, this block simplifies the computations by using x to approximate $\sin(x)$. The baseband PLL model depends on the amplitude of the incoming signal but does not depend on a carrier frequency.

This PLL has these three components:

- An integrator used as a phase detector.
- A filter. You specify the filter's transfer function using the **Lowpass filter numerator** and **Lowpass filter denominator** mask parameters. Each is a vector that gives the respective polynomial's coefficients in order of descending powers of s .

To design a filter, you can use functions such as `butter`, `cheby1`, and `cheby2` in the Signal Processing Toolbox. The default filter is a Chebyshev type II filter whose transfer function arises from the command below.

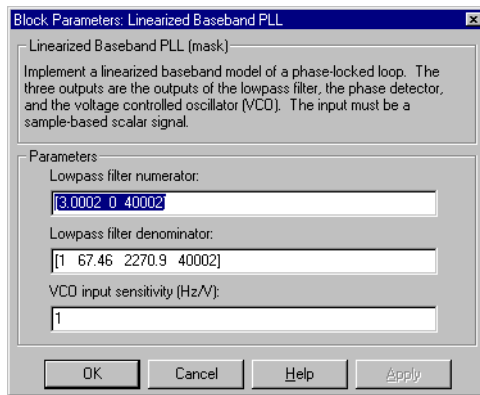
```
[num, den] = cheby2(3, 40, 100, 's')
```

- A voltage-controlled oscillator (VCO). You specify the sensitivity of the VCO signal to its input using the **VCO input sensitivity** parameter. This parameter, measured in Hertz per volt, is a scale factor that determines how much the VCO shifts from its quiescent frequency.

The input signal represents the received signal. The input must be a sample-based scalar signal. The three output ports produce:

- The output of the filter
- The output of the phase detector
- The output of the VCO

Dialog Box



Lowpass filter numerator

The numerator of the lowpass filter's transfer function, represented as a vector that lists the coefficients in order of descending powers of s .

Lowpass filter denominator

The denominator of the lowpass filter's transfer function, represented as a vector that lists the coefficients in order of descending powers of s .

VCO input sensitivity (Hz/V)

This value scales the input to the VCO and, consequently, the shift from the VCO's quiescent frequency.

See Also

Baseband PLL, Phase-Locked Loop

References

For more information about phase-locked loops, see the works listed in "Selected Bibliography for Synchronization" on page 2-92.

Matrix Deinterleaver

Purpose Permute input symbols by filling a matrix by columns and emptying it by rows

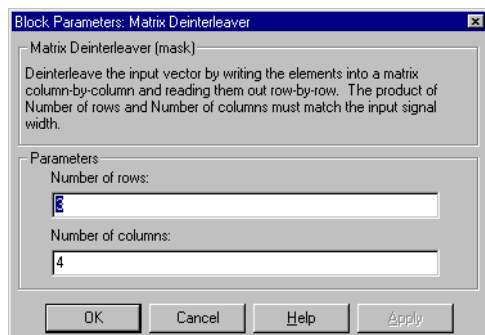
Library Block sublibrary of Interleaving

Description The Matrix Deinterleaver block performs block deinterleaving by filling a matrix with the input symbols column by column and then sending the matrix contents to the output port row by row. The **Number of rows** and **Number of columns** parameters are the dimensions of the matrix that the block uses internally for its computations.



The length of the input vector must be **Number of rows** times **Number of columns**. If the input is frame-based, then it must be a column vector.

Dialog Box



Number of rows

The number of rows in the matrix that the block uses for its computations.

Number of columns

The number of columns in the matrix that the block uses for its computations.

Examples If the **Number of rows** and **Number of columns** parameters are 2 and 3, respectively, then the deinterleaver uses a 2-by-3 matrix for its internal computations. Given an input signal of [1; 2; 3; 4; 5; 6], the block produces an output of [1; 3; 5; 2; 4; 6].

Pair Block Matrix Interleaver

See Also General Block Deinterleaver

Purpose Restore ordering of input symbols by filling a matrix along diagonals

Library Block sublibrary of Interleaving

Description



The Matrix Helical Scan Deinterleaver block performs block deinterleaving by filling a matrix with the input symbols in a helical fashion and then sending the matrix contents to the output port row by row. The **Number of rows** and **Number of columns** parameters are the dimensions of the matrix that the block uses internally for its computations.

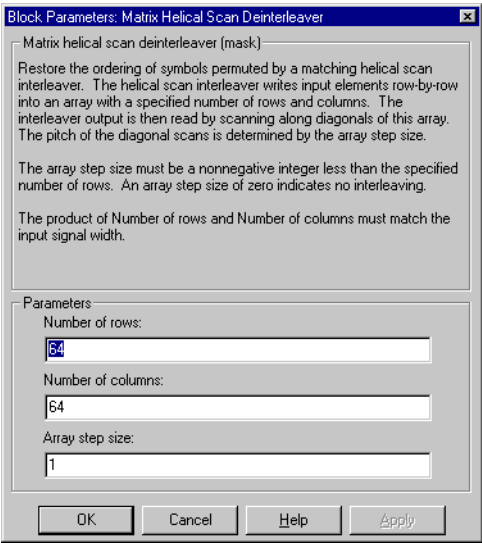
Helical fashion means that the block places input symbols along diagonals of the matrix. The number of elements in each diagonal matches the **Number of columns** parameter, after the block wraps past the edges of the matrix when necessary. The block traverses diagonals so that the row index and column index both increase. Each diagonal after the first one begins one row below the first element of the previous diagonal.

The **Array step size** parameter is the slope of each diagonal, that is, the amount by which the row index increases as the column index increases by one. This parameter must be an integer between zero and the **Number of rows** parameter. If the **Array step size** parameter is zero, then the block does not deinterleave and the output is the same as the input.

The number of elements of the input vector must be the product of **Number of rows** and **Number of columns**. If the input is frame-based, then it must be a column vector.

Matrix Helical Scan Deinterleaver

Dialog Box



Number of rows

The number of rows in the matrix that the block uses for its computations.

Number of columns

The number of columns in the matrix that the block uses for its computations.

Array step size

The slope of the diagonals that the block writes.

Pair Block

Matrix Helical Scan Interleaver

See Also

General Block Deinterleaver

Purpose Permute input symbols by selecting matrix elements along diagonals

Library Block sublibrary of Interleaving

Description



The Matrix Helical Scan Interleaver block performs block interleaving by filling a matrix with the input symbols row by row and then sending the matrix contents to the output port in a helical fashion. The **Number of rows** and **Number of columns** parameters are the dimensions of the matrix that the block uses internally for its computations.

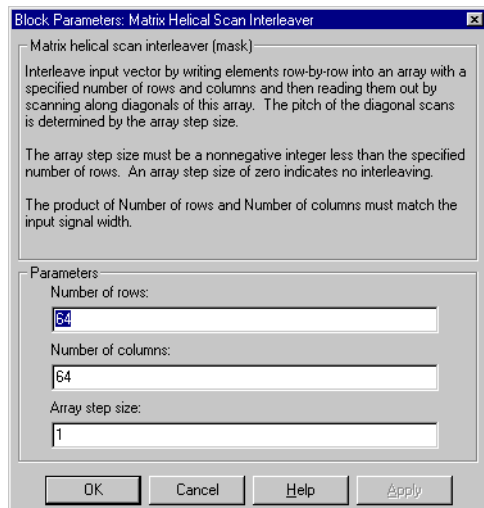
Helical fashion means that the block selects output symbols by selecting elements along diagonals of the matrix. The number of elements in each diagonal matches the **Number of columns** parameter, after the block wraps past the edges of the matrix when necessary. The block traverses diagonals so that the row index and column index both increase. Each diagonal after the first one begins one row below the first element of the previous diagonal.

The **Array step size** parameter is the slope of each diagonal, that is, the amount by which the row index increases as the column index increases by one. This parameter must be an integer between zero and the **Number of rows** parameter. If the **Array step size** parameter is zero, then the block does not interleave and the output is the same as the input.

The number of elements of the input vector must be the product of **Number of rows** and **Number of columns**. If the input is frame-based, then it must be a column vector.

Matrix Helical Scan Interleaver

Dialog Box



Number of rows

The number of rows in the matrix that the block uses for its computations.

Number of columns

The number of columns in the matrix that the block uses for its computations.

Array step size

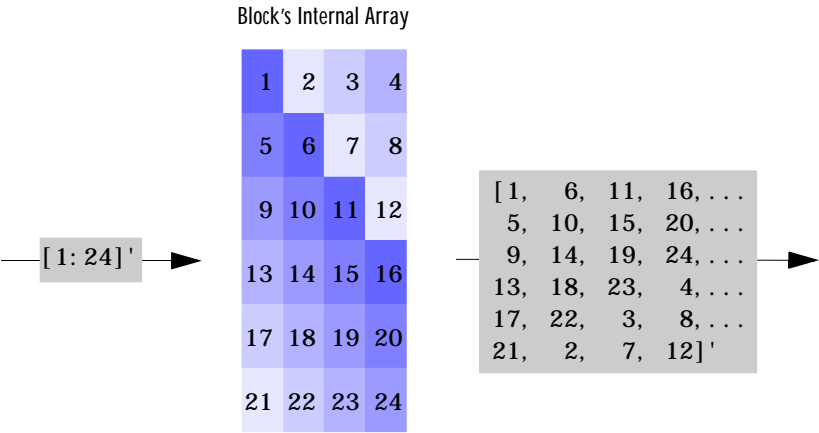
The slope of the diagonals that the block reads.

Examples

If the **Number of rows** and **Number of columns** parameters are 6 and 4, respectively, then the interleaver uses a 6-by-4 matrix for its internal computations. If the **Array step size** parameter is 1, then the diagonals are as shown in the figure below. Positions with the same color form part of the same diagonal, and diagonals with darker colors precede those with lighter colors in the output signal.

Given an input signal of $[1:24]'$, the block produces an output of

```
[1; 6; 11; 16; 5; 10; 15; 20; 9; 14; 19; 24; 13; 18; 23; ...  
4; 17; 22; 3; 8; 21; 2; 7; 12]
```

Pair Block

Matrix Helical Scan Deinterleaver

See Also

General Block Interleaver

Matrix Interleaver

Purpose Permute input symbols by filling a matrix by rows and emptying it by columns

Library Block sublibrary of Interleaving

Description

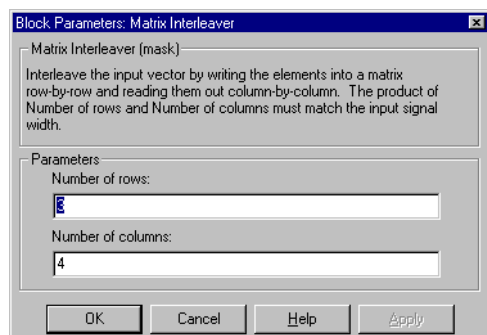


The Matrix Interleaver block performs block interleaving by filling a matrix with the input symbols row by row and then sending the matrix contents to the output port column by column.

The **Number of rows** and **Number of columns** parameters are the dimensions of the matrix that the block uses internally for its computations.

The number of elements of the input vector must be the product of **Number of rows** and **Number of columns**. If the input is frame-based, then it must be a column vector.

Dialog Box



Number of rows

The number of rows in the matrix that the block uses for its computations.

Number of columns

The number of columns in the matrix that the block uses for its computations.

Examples

If the **Number of rows** and **Number of columns** parameters are 2 and 3, respectively, then the interleaver uses a 2-by-3 matrix for its internal computations. Given an input signal of [1; 2; 3; 4; 5; 6], the block produces an output of [1; 4; 2; 5; 3; 6].

Pair Block Matrix Deinterleaver

See Also General Block Interleaver

M-DPSK Demodulator Baseband

Purpose Demodulate DPSK-modulated data

Library PM, in Digital Baseband sublibrary of Modulation

Description



The M-DPSK Demodulator Baseband block demodulates a signal that was modulated using the M-ary differential phase shift keying method. The input is a baseband representation of the modulated signal. The input and output for this block are discrete-time signals. The input can be either a scalar or a frame-based column vector.

The **M-ary number** parameter, M, is the number of possible output symbols that can immediately follow a given output symbol. The block compares the current symbol to the previous symbol. The block's first output is the initial condition of zero (or a group of zeros, if the **Output type** parameter is set to **Bit**) because there is no previous symbol.

Binary or Integer Outputs

If the **Output type** parameter is set to **Integer**, then the block maps a phase difference of

$$\theta + 2\pi m/M$$

to m, where θ is the **Phase offset** parameter and m is an integer between 0 and M-1.

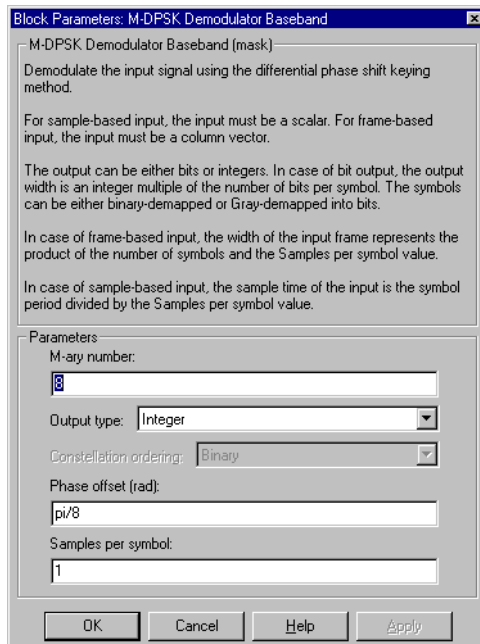
If the **Output type** parameter is set to **Bit** and the **M-ary number** parameter has the form 2^K for some positive integer K, then the block outputs binary representations of integers between 0 and M-1. It outputs a group of K bits, called a binary *word*, for each symbol.

In binary output mode, the **Constellation ordering** parameter indicates how the block maps an integer to a corresponding group of K output bits. See the reference pages for the M-DPSK Modulator Baseband and M-PSK Modulator Baseband blocks for details.

Processing an Upsampled Modulated Signal

The input signal can be an upsampled version of the modulated signal. The **Samples per symbol** parameter is the upsampling factor. If it is greater than 1, then the demodulated signal is delayed by one output sample. For more information, see “Upsampled Signals and Rate Changes” on page 2-72.

Dialog Box



M-ary number

The number of possible modulated symbols that can immediately follow a given symbol.

Output type

Determines whether the output consists of integers or groups of bits.

Constellation ordering

Determines how the block maps each integer to a group of output bits. This field is active only when **Output type** is set to **Bit**.

Phase offset (rad)

The phase difference between the previous and current modulated symbols when the input is zero.

Samples per symbol

The number of input samples that represent each modulated symbol.

M-DPSK Demodulator Baseband

Pair Block M-DPSK Modulator Passband

See Also DBPSK Demodulator Baseband, DQPSK Demodulator Baseband, M-PSK Demodulator Baseband

References [1] Pawula, R. F. "On M-ary DPSK Transmission Over Terrestrial and Satellite Channels." *IEEE Transactions on Communications*, vol. COM-32, July 1984. 752-761.

Purpose	Demodulate DPSK-modulated data
Library	PM, in Digital Passband sublibrary of Modulation

Description



The M-DPSK Demodulator Passband block demodulates a signal that was modulated using the M-ary differential phase shift keying method. The input is a passband representation of the modulated signal. The input and output for this block are discrete-time signals. The input must be a sample-based scalar signal.

The **M-ary number** parameter, M, is the number of possible output symbols that can immediately follow a given output symbol. The block compares the current symbol to the previous symbol. The block's first output is the initial condition of zero because there is no previous symbol.

This block converts the input to an equivalent baseband representation and then uses the baseband equivalent block, M-DPSK Demodulator Baseband, for internal computations. The following parameters in this block are the same as those of the baseband equivalent block:

- **M-ary number**
- **Output type**
- **Constellation ordering**

Parameters Specific to Passband Simulation

Passband simulation uses a carrier signal. The **Carrier frequency** and **Carrier initial phase** parameters specify the frequency and initial phase, respectively, of the carrier signal. The **Input sample time** parameter specifies the sample time of the input signal, while the **Symbol period** parameter equals the sample time of the output signal.

This block uses a baseband representation of the modulated signal as an intermediate signal during internal computations. The **Baseband samples per symbol** parameter indicates how many baseband samples correspond to each integer or binary word in the output.

The timing-related parameters must satisfy these relationships:

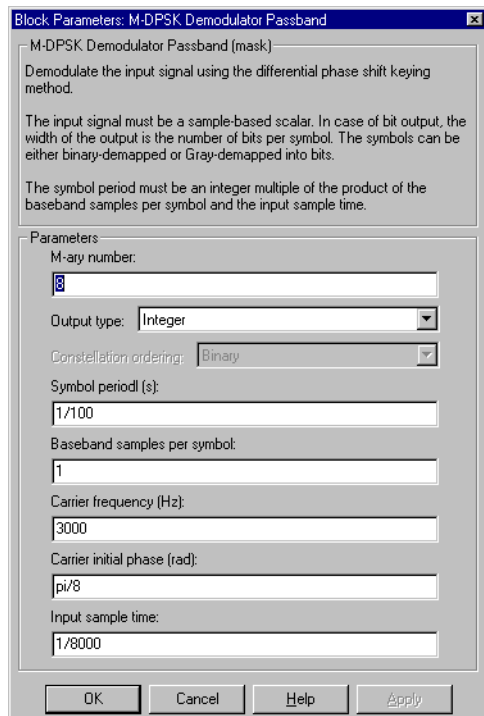
- **Input sample time** > (**Carrier frequency**)⁻¹
- **Symbol period** < [2***Carrier frequency** + 2/(**Input sample time**)]⁻¹

M-DPSK Demodulator Passband

- **Input sample time** = $K \times \text{Symbol period} \times \text{Baseband samples per symbol}$ for some integer K

Also, this block incurs an extra output period of delay compared to its baseband equivalent block.

Dialog Box



The dialog box is titled "Block Parameters: M-DPSK Demodulator Passband". It contains the following text and controls:

M-DPSK Demodulator Passband (mask)
Demodulate the input signal using the differential phase shift keying method.

The input signal must be a sample-based scalar. In case of bit output, the width of the output is the number of bits per symbol. The symbols can be either binary-demapped or Gray-demapped into bits.

The symbol period must be an integer multiple of the product of the baseband samples per symbol and the input sample time.

Parameters

M-ary number:

Output type:

Constellation ordering:

Symbol period (s):

Baseband samples per symbol:

Carrier frequency (Hz):

Carrier initial phase (rad):

Input sample time:

Buttons: OK, Cancel, Help, Apply

M-ary number

The number of possible modulated symbols that can immediately follow a given symbol.

Output type

Determines whether the output consists of integers or groups of bits.

Constellation ordering

Determines how the block maps each integer to a group of output bits. This field is active only when **Output type** is set to **Bit**.

Symbol period (s)

The symbol period, which equals the sample time of the output.

Baseband samples per symbol

The number of baseband samples that represent each modulated symbol, after the block converts the passband input to a baseband intermediary signal.

Carrier frequency (Hz)

The frequency of the carrier.

Carrier initial phase (rad)

The initial phase of the carrier in radians.

Input sample time

The sample time of the input signal.

Pair Block M-DPSK Modulator Passband

See Also M-DPSK Demodulator Baseband

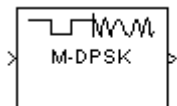
References [1] Pawula, R. F. "On M-ary DPSK Transmission Over Terrestrial and Satellite Channels." *IEEE Transactions on Communications*, vol. COM-32, July 1984. 752-761.

M-DPSK Modulator Baseband

Purpose Modulate using the M-ary differential phase shift keying method

Library PM, in Digital Baseband sublibrary of Modulation

Description



The M-DPSK Modulator Baseband block modulates using the M-ary differential phase shift keying method. The output is a baseband representation of the modulated signal. The **M-ary number** parameter, M, is the number of possible output symbols that can immediately follow a given output symbol.

The input must be a discrete-time signal.

Inputs and Constellation Types

If the **Input type** parameter is set to **Integer**, then valid input values are integers between 0 and M-1. In this case, the input can be either a scalar or a frame-based column vector. If the first input is m, then the modulated symbol is

$$\exp(j\theta + j\pi m/2)$$

where θ is the **Phase offset** parameter. If a successive input is m, then the modulated symbol is the previous modulated symbol multiplied by $\exp(j\theta + j\pi m/2)$.

If the **Input type** parameter is set to **Bit** and the **M-ary number** parameter has the form 2^K for some positive integer K, then the block accepts binary representations of integers between 0 and M-1. It modulates each group of K bits, called a binary *word*. The input can be either a vector of length K or a frame-based column vector whose length is an integer multiple of K.

In binary input mode, the **Constellation ordering** parameter indicates how the block maps a group of K input bits to a corresponding phase difference. The **Binary** option uses a natural binary-to-integer mapping, while the **Gray** option uses a Gray-coded assignment of phase differences. For example, the table

below indicates the assignment of phase difference to three-bit inputs, for both the **Binary** and **Gray** options. θ is the **Phase offset** parameter.

Input	Binary-Coded Phase Differences	Gray-Coded Phase Differences
[0 0 0]	$j\theta$	$j\theta$
[0 0 1]	$j\theta + j\pi/2$	$j\theta + j\pi/2$
[0 1 0]	$j\theta + j\pi 2/2$	$j\theta + j\pi 3/2$
[0 1 1]	$j\theta + j\pi 3/2$	$j\theta + j\pi 2/2$
[1 0 0]	$j\theta + j\pi 4/2$	$j\theta + j\pi 6/2$
[1 0 1]	$j\theta + j\pi 5/2$	$j\theta + j\pi 7/2$
[1 1 0]	$j\theta + j\pi 6/2$	$j\theta + j\pi 5/2$
[1 1 1]	$j\theta + j\pi 7/2$	$j\theta + j\pi 4/2$

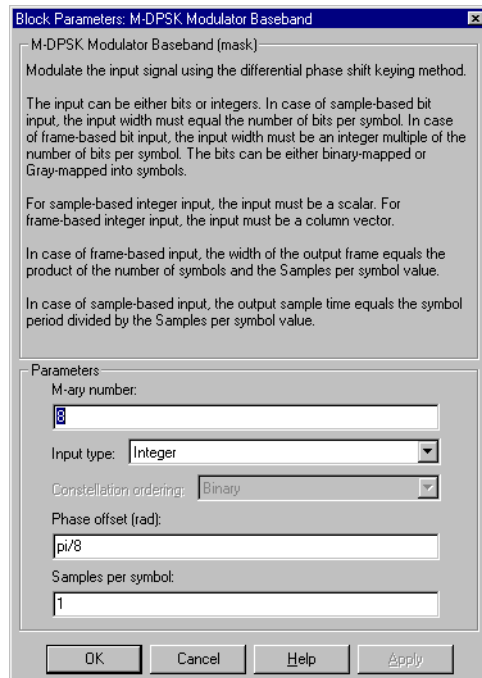
For more details about the **Binary** and **Gray** options, see the reference page for the M-PSK Modulator Baseband block. The signal constellation for that block corresponds to the arrangement of phase differences for this block.

Upsampling the Modulated Signal

This block can output an upsampled version of the modulated signal. The **Samples per symbol** parameter is the upsampling factor. It must be a positive integer. For more information, see “Upsampled Signals and Rate Changes” on page 2-72.

M-DPSK Modulator Baseband

Dialog Box



M-ary number

The number of possible output symbols that can immediately follow a given output symbol.

Input type

Indicates whether the input consists of integers or groups of bits. If this parameter is set to **Bit**, then the **M-ary number** parameter must be 2^K for some positive integer K.

Constellation ordering

Determines how the block maps each group of input bits to a corresponding integer. This field is active only when **Input type** is set to **Bit**.

Phase offset (rad)

The phase difference between the previous and current modulated symbols when the input is zero.

Samples per symbol

The number of output samples that the block produces for each integer or binary word in the input.

Pair Block M-DPSK Demodulator Baseband

See Also DBPSK Modulator Baseband, DQPSK Modulator Baseband, M-PSK Modulator Baseband

References [1] Pawula, R. F. "On M-ary DPSK Transmission Over Terrestrial and Satellite Channels." *IEEE Transactions on Communications*, vol. COM-32, July 1984. 752-761.

M-DPSK Modulator Passband

Purpose Modulate using the M-ary differential phase shift keying method

Library PM, in Digital Passband sublibrary of Modulation

Description



The M-DPSK Modulator Passband block modulates using the M-ary differential phase shift keying method. The output is a passband representation of the modulated signal. The **M-ary number** parameter, M, is the number of possible output symbols that can immediately follow a given output symbol.

This block uses the baseband equivalent block, M-DPSK Modulator Baseband, for internal computations and converts the resulting baseband signal to a passband representation. The following parameters in this block are the same as those of the baseband equivalent block:

- **M-ary number**
- **Input type**
- **Constellation ordering**

The input must be sample-based. If the **Input type** parameter is **Bit**, then the input must be a vector of length $\log_2(M)$. If the **Input type** parameter is **Integer**, then the input must be a scalar.

Parameters Specific to Passband Simulation

Passband simulation uses a carrier signal. The **Carrier frequency** and **Carrier initial phase** parameters specify the frequency and initial phase, respectively, of the carrier signal. The **Symbol period** parameter must equal the sample time of the input signal, while the **Output sample time** parameter determines the sample time of the output signal.

This block uses a baseband representation of the modulated signal as an intermediate result during internal computations. The **Baseband samples per symbol** parameter indicates how many baseband samples correspond to each integer or binary word in the input, before the block converts them to a passband output.

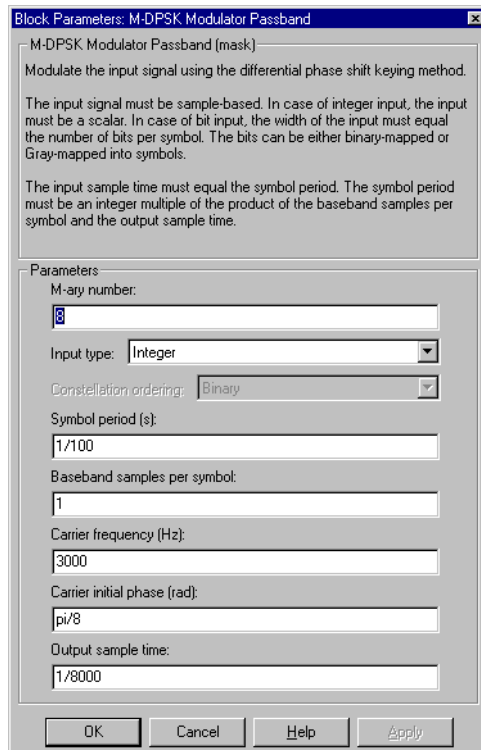
The timing-related parameters must satisfy these relationships:

- **Symbol period** > (**Carrier frequency**)⁻¹
- **Output sample time** < [2***Carrier frequency** + 2/(**Symbol period**)]⁻¹

- **Symbol period** = $K \times \text{Output sample time} \times \text{Baseband samples per symbol}$ for some integer K

Furthermore, **Carrier frequency** is typically much larger than the highest frequency of the unmodulated signal.

Dialog Box



The dialog box is titled "Block Parameters: M-DPSK Modulator Passband". It contains a description of the block's function and a set of parameters.

M-DPSK Modulator Passband (mask)
Modulate the input signal using the differential phase shift keying method.

The input signal must be sample-based. In case of integer input, the input must be a scalar. In case of bit input, the width of the input must equal the number of bits per symbol. The bits can be either binary-mapped or Gray-mapped into symbols.

The input sample time must equal the symbol period. The symbol period must be an integer multiple of the product of the baseband samples per symbol and the output sample time.

Parameters

M-ary number:

Input type:

Constellation ordering:

Symbol period (s):

Baseband samples per symbol:

Carrier frequency (Hz):

Carrier initial phase (rad):

Output sample time:

Buttons: OK, Cancel, Help, Apply

M-ary number

The number of possible output symbols that can immediately follow a given output symbol.

Input type

Indicates whether the input consists of integers or groups of bits. If this parameter is set to **Bit**, then the **M-ary number** parameter must be 2^K for some positive integer K .

M-DPSK Modulator Passband

Constellation ordering

Determines how the block maps each group of input bits to a corresponding integer. This field is active only when **Input type** is set to **Bit**.

Symbol period (s)

The symbol period, which must equal the sample time of the input.

Baseband samples per symbol

The number of baseband samples that correspond to each integer or binary word in the input, before the block converts them to a passband output.

Carrier frequency (Hz)

The frequency of the carrier.

Carrier initial phase (rad)

The initial phase of the carrier in radians.

Output sample time

The sample time of the output signal.

Pair Block M-DPSK Demodulator Passband

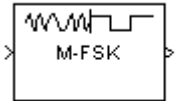
See Also M-DPSK Modulator Baseband

References [1] Pawula, R. F. "On M-ary DPSK Transmission Over Terrestrial and Satellite Channels." *IEEE Transactions on Communications*, vol. COM-32, July 1984. 752-761.

Purpose Demodulate FSK-modulated data

Library FM, in Digital Baseband sublibrary of Modulation

Description



The M-FSK Demodulator Baseband block demodulates a signal that was modulated using the M-ary frequency shift keying method. The input is a baseband representation of the modulated signal. The input and output for this block are discrete-time signals. The input can be either a scalar or a frame-based column vector.

The **M-ary number** parameter, M , is the number of frequencies in the modulated signal. The **Frequency separation** parameter is the distance, in Hz, between successive frequencies of the modulated signal.

Binary or Integer Outputs

If the **Output type** parameter is set to **Integer**, then the block outputs integers between 0 and $M-1$.

If the **Output type** parameter is set to **Bit** and the **M-ary number** parameter has the form 2^K for some positive integer K , then the block outputs binary representations of integers between 0 and $M-1$. It outputs a group of K bits, called a binary *word*, for each symbol.

In binary output mode, the **Symbol set ordering** parameter indicates how the block maps an integer to a corresponding group of K output bits. See the reference pages for the M-FSK Modulator Baseband and M-PSK Modulator Baseband blocks for details.

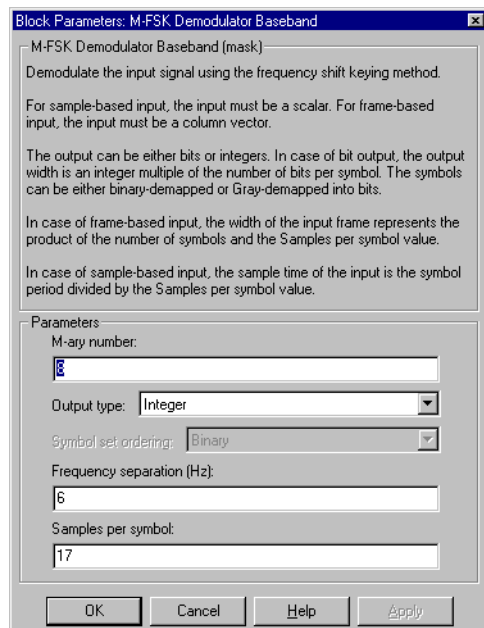
Whether the output is an integer or a binary representation of an integer, the block maps the highest frequency to the integer 0 and maps the lowest frequency to the integer $M-1$. In baseband simulation, the lowest frequency is the negative frequency with the largest absolute value.

Processing an Upsampled Modulated Signal

The input signal can be an upsampled version of the modulated signal. The **Samples per symbol** parameter is the upsampling factor. It must be a positive integer. For more information, see “Upsampled Signals and Rate Changes” on page 2-72.

M-FSK Demodulator Baseband

Dialog Box



M-ary number

The number of frequencies in the modulated signal.

Output type

Determines whether the output consists of integers or groups of bits. If this parameter is set to **Bit**, then the **M-ary number** parameter must be 2^K for some positive integer K .

Symbol set ordering

Determines how the block maps each integer to a group of output bits. This field is active only when **Output type** is set to **Bit**.

Frequency separation (Hz)

The distance between successive frequencies in the modulated signal.

Samples per symbol

The number of input samples that represent each modulated symbol.

Pair Block M-FSK Modulator Baseband

See Also CPFSK Demodulator Baseband

M-FSK Demodulator Passband

Purpose Modulate using the M-ary frequency shift keying method

Library FM, in Digital Passband sublibrary of Modulation

Description



The M-FSK Demodulator Passband block demodulates a signal that was modulated using the M-ary frequency shift keying method. The input is a passband representation of the modulated signal. The **M-ary number** parameter, M, is the number of frequencies in the modulated signal.

This block converts the input to an equivalent baseband representation and then uses the baseband equivalent block, M-FSK Demodulator Baseband, for internal computations. The following parameters in this block are the same as those of the baseband equivalent block:

- **M-ary number**
- **Output type**
- **Signal set ordering**
- **Frequency separation**

The input must be a sample-based scalar signal.

Parameters Specific to Passband Simulation

Passband simulation uses a carrier signal. The **Carrier frequency** and **Carrier initial phase** parameters specify the frequency and initial phase, respectively, of the carrier signal. The **Input sample time** parameter specifies the sample time of the input signal, while the **Symbol period** parameter equals the sample time of the output signal.

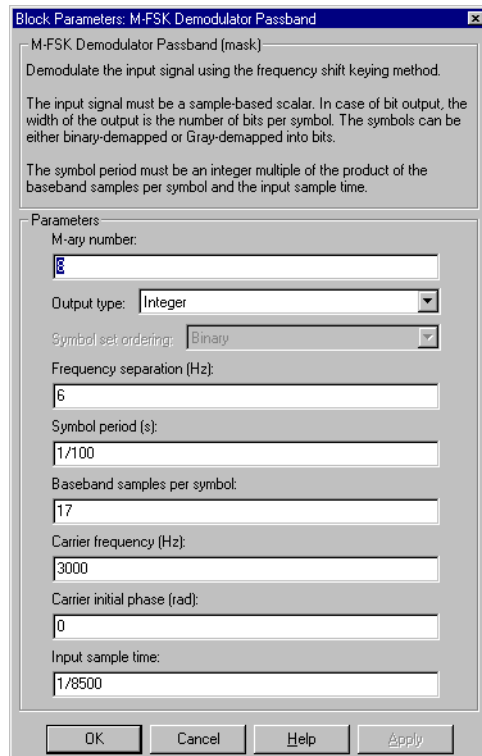
This block uses a baseband representation of the modulated signal as an intermediate signal during internal computations. The **Baseband samples per symbol** parameter indicates how many baseband samples correspond to each integer or binary word in the output.

The timing-related parameters must satisfy these relationships:

- **Input sample time** > $(\text{Carrier frequency})^{-1}$
- **Symbol period** < $[2 * \text{Carrier frequency} + 2 / (\text{Input sample time})]^{-1}$
- **Input sample time** = $K * \text{Symbol period} * \text{Baseband samples per symbol}$ for some integer K

Also, this block incurs an extra output period of delay compared to its baseband equivalent block.

Dialog Box



The dialog box is titled "Block Parameters: M-FSK Demodulator Passband". It contains a description of the block's function and a list of parameters to be configured.

M-FSK Demodulator Passband (mask)
Demodulate the input signal using the frequency shift keying method.

The input signal must be a sample-based scalar. In case of bit output, the width of the output is the number of bits per symbol. The symbols can be either binary-demapped or Gray-demapped into bits.

The symbol period must be an integer multiple of the product of the baseband samples per symbol and the input sample time.

Parameters

- M-ary number: 8
- Output type: Integer
- Symbol set ordering: Binary
- Frequency separation (Hz): 6
- Symbol period (s): 1/100
- Baseband samples per symbol: 17
- Carrier frequency (Hz): 3000
- Carrier initial phase (rad): 0
- Input sample time: 1/8500

Buttons: OK, Cancel, Help, Apply

M-ary number

The number of frequencies in the modulated signal.

Output type

Determines whether the output consists of integers or groups of bits. If this parameter is set to **Bit**, then the **M-ary number** parameter must be 2^K for some positive integer K .

Symbol set ordering

Determines how the block maps each integer to a group of output bits. This field is active only when **Output type** is set to **Bit**.

M-FSK Demodulator Passband

Frequency separation (Hz)

The distance between successive frequencies in the modulated signal.

Symbol period (s)

The symbol period, which equals the sample time of the output.

Baseband samples per symbol

The number of baseband samples that represent each modulated symbol, after the block converts the passband input to a baseband intermediary signal.

Carrier frequency (Hz)

The frequency of the carrier.

Carrier initial phase (rad)

The initial phase of the carrier in radians.

Input sample time

The sample time of the input signal.

Pair Block

M-FSK Modulator Passband

See Also

M-FSK Demodulator Baseband, CPFSK Demodulator Passband

Purpose Modulate using the M-ary frequency shift keying method

Library FM, in Digital Baseband sublibrary of Modulation

Description



The M-FSK Modulator Baseband block modulates using the M-ary frequency shift keying method. The output is a baseband representation of the modulated signal.

The **M-ary number** parameter, M , is the number of frequencies in the modulated signal. The **Frequency separation** parameter is the distance, in Hz, between successive frequencies of the modulated signal. If the **Phase continuity** parameter is set to **Continuous**, then the modulated signal maintains its phase even when it changes its frequency. If the **Phase continuity** parameter is set to **Discontinuous**, then the modulated signal comprises portions of M sinusoids of different frequencies; thus, a change in the input value might cause a change in the phase of the modulated signal.

Input Signal Values

The input and output for this block are discrete-time signals. The **Input type** parameter determines whether the block accepts integers between 0 and $M-1$, or binary representations of integers:

- If **Input type** is set to **Integer**, then the block accepts integers. The input can be either a scalar or a frame-based column vector.
- If **Input type** is set to **Bit**, then the block accepts groups of K bits, called binary words. The input can be either a vector of length K or a frame-based column vector whose length is an integer multiple of K . The **Symbol set ordering** parameter indicates how the block assigns binary words to corresponding integers.
 - If **Symbol set ordering** is set to **Binary**, then the block uses a natural binary-coded ordering.
 - If **Symbol set ordering** is set to **Gray**, then the block uses a Gray-coded ordering. For details about the Gray coding, see the reference page for the M-PSK Modulator Baseband block.

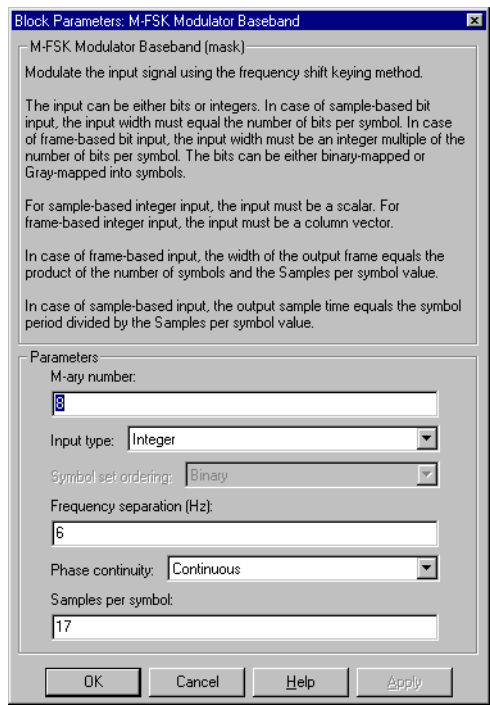
Whether the input is an integer or a binary representation of an integer, the block maps the integer 0 to the highest frequency and maps the integer $M-1$ to the lowest frequency. In baseband simulation, the lowest frequency is the negative frequency with the largest absolute value.

M-FSK Modulator Baseband

Upsampling the Modulated Signal

This block can output an upsampled version of the modulated signal. The **Samples per symbol** parameter is the upsampling factor. It must be a positive integer. For more information, see “Upsampled Signals and Rate Changes” on page 2-72.

Dialog Box



M-ary number

The number of frequencies in the modulated signal.

Input type

Indicates whether the input consists of integers or groups of bits. If this parameter is set to **Bit**, then the **M-ary number** parameter must be 2^K for some positive integer K.

Symbol set ordering

Determines how the block maps each group of input bits to a corresponding integer. This field is active only when **Input type** is set to **Bit**.

Frequency separation (Hz)

The distance between successive frequencies in the modulated signal.

Phase continuity

Determines whether the modulated signal changes phases in a continuous or discontinuous way.

Samples per symbol

The number of output samples that the block produces for each integer or binary word in the input.

Pair Block M-FSK Demodulator Baseband

See Also CPFSK Modulator Baseband

M-FSK Modulator Passband

Purpose Modulate using the M-ary frequency shift keying method

Library FM, in Digital Passband sublibrary of Modulation

Description



The M-FSK Modulator Passband block modulates using the M-ary frequency shift keying method. The output is a passband representation of the modulated signal. The **M-ary number** parameter, M, is the number of frequencies in the modulated signal.

This block uses the baseband equivalent block, M-FSK Modulator Baseband, for internal computations and converts the resulting baseband signal to a passband representation. The following parameters in this block are the same as those of the baseband equivalent block:

- **M-ary number**
- **Input type**
- **Symbol set ordering**
- **Frequency separation**
- **Phase continuity**

The input must be sample-based. If the **Input type** parameter is **Bit**, then the input must be a vector of length $\log_2(M)$. If the **Input type** parameter is **Integer**, then the input must be a scalar.

Whether the input is an integer or a binary representation of an integer, the block maps the integer 0 to the highest frequency and maps the integer M-1 to the lowest frequency.

Parameters Specific to Passband Simulation

Passband simulation uses a carrier signal. The **Carrier frequency** and **Carrier initial phase** parameters specify the frequency and initial phase, respectively, of the carrier signal. The **Symbol period** parameter must equal the sample time of the input signal, while the **Output sample time** parameter determines the sample time of the output signal.

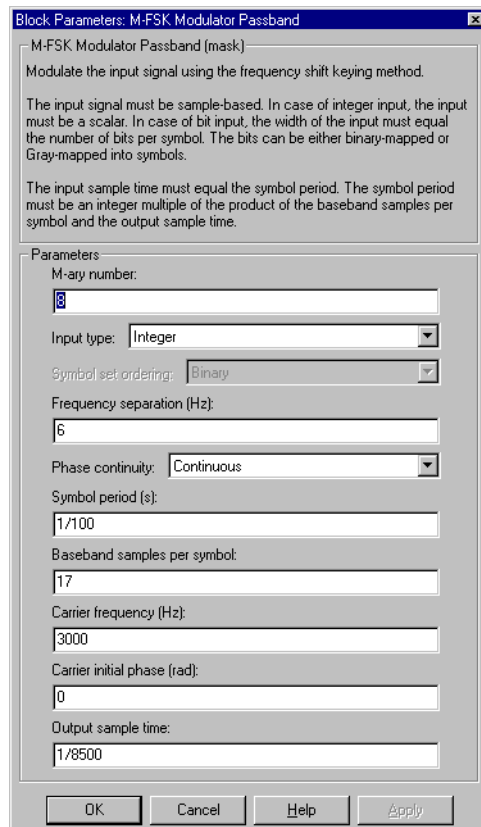
This block uses a baseband representation of the modulated signal as an intermediate result during internal computations. The **Baseband samples per symbol** parameter indicates how many baseband samples correspond to each integer or binary word in the input, before the block converts them to a passband output.

The timing-related parameters must satisfy these relationships:

- **Symbol period** > **(Carrier frequency)**⁻¹
- **Output sample time** < [2***Carrier frequency** + 2/(**Symbol period**)]⁻¹
- **Symbol period** = **K*Output sample time*Baseband samples per symbol**
for some integer K

Furthermore, **Carrier frequency** is typically much larger than the highest frequency of the unmodulated signal.

Dialog Box



Block Parameters: M-FSK Modulator Passband

M-FSK Modulator Passband (mask)
Modulate the input signal using the frequency shift keying method.

The input signal must be sample-based. In case of integer input, the input must be a scalar. In case of bit input, the width of the input must equal the number of bits per symbol. The bits can be either binary-mapped or Gray-mapped into symbols.

The input sample time must equal the symbol period. The symbol period must be an integer multiple of the product of the baseband samples per symbol and the output sample time.

Parameters

M-ary number:

Input type:

Symbol set ordering:

Frequency separation (Hz):

Phase continuity:

Symbol period (s):

Baseband samples per symbol:

Carrier frequency (Hz):

Carrier initial phase (rad):

Output sample time:

OK Cancel Help Apply

M-ary number

The number of frequencies in the modulated signal.

M-FSK Modulator Passband

Input type

Indicates whether the input consists of integers or groups of bits. If this parameter is set to **Bit**, then the **M-ary number** parameter must be 2^K for some positive integer K.

Symbol set ordering

Determines how the block maps each group of input bits to a corresponding integer. This field is active only when **Input type** is set to **Bit**.

Frequency separation (Hz)

The distance between successive frequencies in the modulated signal.

Phase continuity

Determines whether the modulated signal changes phases in a continuous or discontinuous way.

Symbol period (s)

The symbol period, which must equal the sample time of the input.

Baseband samples per symbol

The number of baseband samples that correspond to each integer or binary word in the input, before the block converts them to a passband output.

Carrier frequency (Hz)

The frequency of the carrier.

Carrier initial phase (rad)

The initial phase of the carrier in radians.

Output sample time

The sample time of the output signal.

Pair Block

M-FSK Demodulator Passband

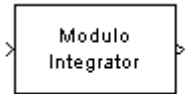
See Also

M-FSK Modulator Baseband, CPFSK Modulator Passband

Purpose Integrate in continuous time and reduce by a modulus

Library Integrators, in Basic Comm Functions

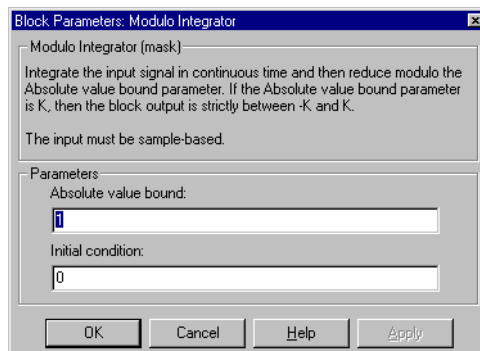
Description The Modulo Integrator block integrates its input signal in continuous time and then reduces modulo the **Absolute value bound** parameter. If the **Absolute value bound** parameter is K, then the block output is strictly between -K and K.



The input must be sample-based. The block processes each vector element independently.

This block's functionality is useful for monotonically increasing or decreasing functions, but works with any integrable function. This block uses the Forward Euler integration method.

Dialog Box



Absolute value bound

The modulus by which the integration result is reduced. This parameter must be nonzero.

Initial condition

The initial condition for integration.

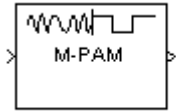
See Also Discrete Modulo Integrator, Integrator (Simulink)

M-PAM Demodulator Baseband

Purpose Demodulate PAM-modulated data

Library AM, in Digital Baseband sublibrary of Modulation

Description



The M-PAM Demodulator Baseband block demodulates a signal that was modulated using the M-ary pulse amplitude modulation. The input is a baseband representation of the modulated signal.

The signal constellation has M points, where M is the **M-ary number** parameter. M must be an even integer. The block scales the signal constellation based on how you set the **Normalization method** parameter. For details on the constellation and its scaling, see the reference page for the M-PAM Modulator Baseband block.

The input can be either a scalar or a frame-based column vector.

Output Signal Values

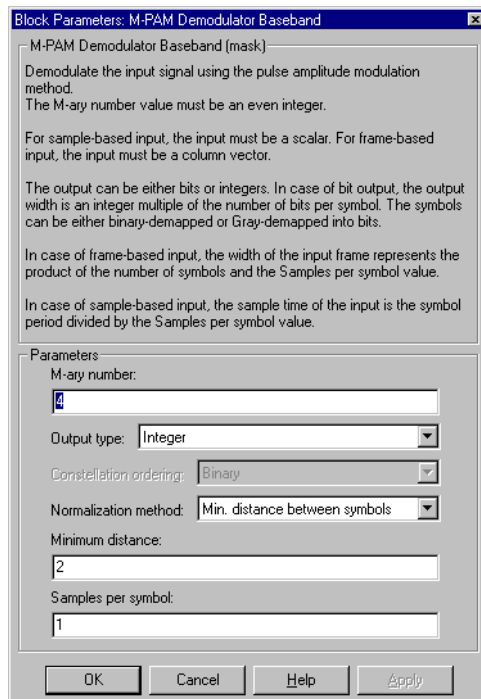
The **Output type** parameter determines whether the block produces integers or binary representations of integers. If **Output type** is set to **Integer**, then the block produces integers. If **Output type** is set to **Bit**, then the block produces a group of K bits, called a binary word, for each symbol. The **Constellation ordering** parameter indicates how the block assigns binary words to points of the signal constellation. More details are on the reference page for the M-PAM Modulator Baseband block.

Processing an Upsampled Modulated Signal

The input signal can be an upsampled version of the modulated signal. The **Samples per symbol** parameter is the upsampling factor. It must be a positive integer.

For more information, see “Upsampled Signals and Rate Changes” on page 2-72.

Dialog Box



M-ary number

The number of points in the signal constellation. It must be an even integer.

Output type

Determines whether the output consists of integers or groups of bits. If this parameter is set to **Bit**, then the **M-ary number** parameter must be 2^K for some positive integer K .

Constellation ordering

Determines how the block maps each integer to a group of output bits. This field is active only when **Output type** is set to **Bit**.

Normalization method

Determines how the block scales the signal constellation. Choices are **Min. distance between symbols**, **Average Power**, and **Peak Power**.

M-PAM Demodulator Baseband

Minimum distance

The distance between two nearest constellation points. This field appears only when **Normalization method** is set to **Min. distance between symbols**.

Average power (watts)

The average power of the symbols in the constellation. This field appears only when **Normalization method** is set to **Average Power**.

Peak power (watts)

The maximum power of the symbols in the constellation. This field appears only when **Normalization method** is set to **Peak Power**.

Samples per symbol

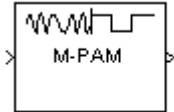
The number of input samples that represent each modulated symbol.

Pair Block	M-PAM Modulator Baseband
See Also	General QAM Demodulator Baseband

Purpose Demodulate PAM-modulated data

Library AM, in Digital Passband sublibrary of Modulation

Description



The M-PAM Demodulator Passband block demodulates a signal that was modulated using M-ary pulse amplitude modulation. The input is a passband representation of the modulated signal. The input must be a sample-based scalar signal.

This block converts the input to an equivalent baseband representation and then uses the baseband equivalent block, M-PAM Demodulator Baseband, for internal computations. The following parameters in this block are the same as those of the baseband equivalent block:

- **M-ary number**
- **Output type**
- **Constellation ordering**
- **Normalization method**
- **Minimum distance**
- **Average power**
- **Peak power**

Parameters Specific to Passband Simulation

Passband simulation uses a carrier signal. The **Carrier frequency** and **Carrier initial phase** parameters specify the frequency and initial phase, respectively, of the carrier signal. The **Input sample time** parameter specifies the sample time of the input signal, while the **Symbol period** parameter equals the sample time of the output signal.

This block uses a baseband representation of the modulated signal as an intermediate signal during internal computations. The **Baseband samples per symbol** parameter indicates how many baseband samples correspond to each integer or binary word in the output.

The timing-related parameters must satisfy these relationships:

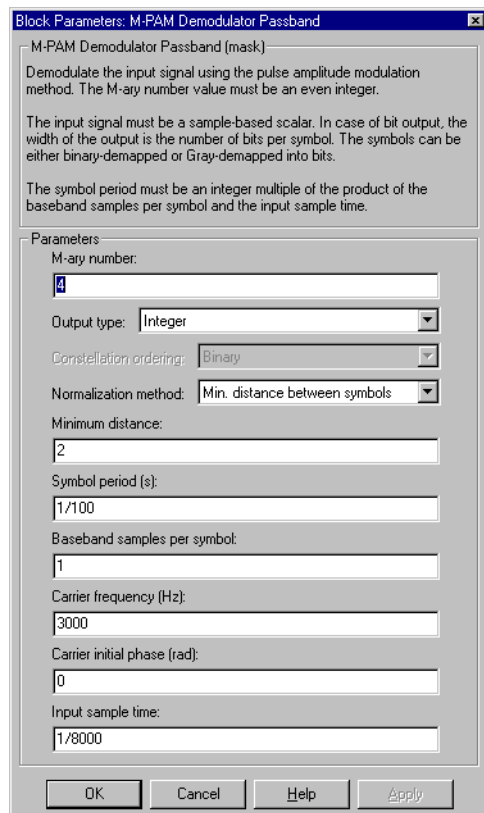
- **Input sample time** > (**Carrier frequency**)⁻¹
- **Symbol period** < [2***Carrier frequency** + 2/(**Input sample time**)]⁻¹

M-PAM Demodulator Passband

- **Input sample time** = $K \times \text{Symbol period} \times \text{Baseband samples per symbol}$ for some integer K

Also, this block incurs an extra output period of delay compared to its baseband equivalent block.

Dialog Box



The dialog box is titled "Block Parameters: M-PAM Demodulator Passband". It contains a description of the block's function and a section for parameters.

M-PAM Demodulator Passband (mask)

Demodulate the input signal using the pulse amplitude modulation method. The M-ary number value must be an even integer.

The input signal must be a sample-based scalar. In case of bit output, the width of the output is the number of bits per symbol. The symbols can be either binary-demapped or Gray-demapped into bits.

The symbol period must be an integer multiple of the product of the baseband samples per symbol and the input sample time.

Parameters

M-ary number:

Output type:

Constellation ordering:

Normalization method:

Minimum distance:

Symbol period (s):

Baseband samples per symbol:

Carrier frequency (Hz):

Carrier initial phase (rad):

Input sample time:

Buttons: OK, Cancel, Help, Apply

M-ary number

The number of points in the signal constellation. It must be an even integer.

Output type

Determines whether the output consists of integers or groups of bits. If this parameter is set to **Bit**, then the **M-ary number** parameter must be 2^K for some positive integer K.

Constellation ordering

Determines how the block maps each integer to a group of output bits. This field is active only when **Output type** is set to **Bit**.

Normalization method

Determines how the block scales the signal constellation. Choices are **Min. distance between symbols**, **Average Power**, and **Peak Power**.

Minimum distance

The distance between two nearest constellation points. This field appears only when **Normalization method** is set to **Min. distance between symbols**.

Average power (watts)

The average power of the symbols in the constellation. This field appears only when **Normalization method** is set to **Average Power**.

Peak power (watts)

The maximum power of the symbols in the constellation. This field appears only when **Normalization method** is set to **Peak Power**.

Symbol period (s)

The symbol period, which equals the sample time of the output.

Baseband samples per symbol

The number of baseband samples that represent each modulated symbol, after the block converts the passband input to a baseband intermediary signal.

Carrier frequency (Hz)

The frequency of the carrier.

Carrier initial phase (rad)

The initial phase of the carrier in radians.

M-PAM Demodulator Passband

Input sample time
The sample time of the input signal.

Pair Block M-PAM Modulator Passband

See Also M-PAM Demodulator Baseband

Purpose Modulate using M-ary pulse amplitude modulation

Library AM, in Digital Baseband sublibrary of Modulation

Description



The M-PAM Modulator Baseband block modulates using M-ary pulse amplitude modulation. The output is a baseband representation of the modulated signal. The **M-ary number** parameter, M, is the number of points in the signal constellation. It must be an even integer.

Constellation Size and Scaling

Baseband M-ary pulse amplitude modulation using the block's default signal constellation maps an integer m between 0 and M-1 to the complex value

$$2m - M + 1$$

Note This is actually a real number. The block's output signal is a complex data-type signal whose imaginary part is zero.

The block scales the default signal constellation based on how you set the **Normalization method** parameter. The table below lists the possible scaling conditions.

Value of Normalization method Parameter	Scaling Condition
Min. distance between symbols	The nearest pair of points in the constellation is separated by the value of the Minimum distance parameter
Average Power	The average power of the symbols in the constellation is the Average power parameter
Peak Power	The maximum power of the symbols in the constellation is the Peak power parameter

M-PAM Modulator Baseband

Input Signal Values

The input and output for this block are discrete-time signals. The **Input type** parameter determines whether the block accepts integers between 0 and M-1, or binary representations of integers.

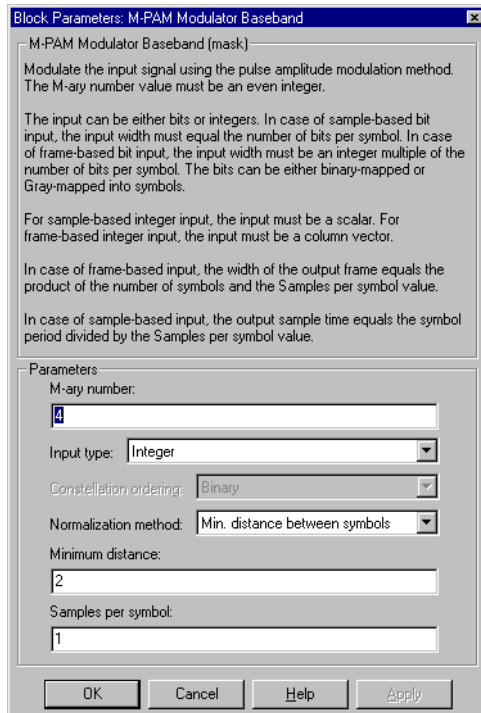
- If **Input type** is set to **Integer**, then the block accepts integers. The input can be either a scalar or a frame-based column vector.
- If **Input type** is set to **Bit**, then the block accepts groups of K bits, called binary words. The input can be either a vector of length K or a frame-based column vector whose length is an integer multiple of K. The **Constellation ordering** parameter indicates how the block assigns binary words to points of the signal constellation.
 - If **Constellation ordering** is set to **Binary**, then the block uses a natural binary-coded constellation.
 - If **Constellation ordering** is set to **Gray**, then the block uses a Gray-coded constellation.

For details about the Gray coding, see the reference page for the M-PSK Modulator Baseband block.

Upsampling the Modulated Signal

This block can output an upsampled version of the modulated signal. The **Samples per symbol** parameter is the upsampling factor. It must be a positive integer. For more information, see “Upsampled Signals and Rate Changes” on page 2-72.

Dialog Box



The dialog box is titled "Block Parameters: M-PAM Modulator Baseband". It contains a text area with the following text: "M-PAM Modulator Baseband (mask)
Modulate the input signal using the pulse amplitude modulation method. The M-ary number value must be an even integer.

The input can be either bits or integers. In case of sample-based bit input, the input width must equal the number of bits per symbol. In case of frame-based bit input, the input width must be an integer multiple of the number of bits per symbol. The bits can be either binary-mapped or Gray-mapped into symbols.

For sample-based integer input, the input must be a scalar. For frame-based integer input, the input must be a column vector.

In case of frame-based input, the width of the output frame equals the product of the number of symbols and the Samples per symbol value.

In case of sample-based input, the output sample time equals the symbol period divided by the Samples per symbol value."

Below the text area is a section labeled "Parameters" with the following fields:

- M-ary number:
- Input type:
- Constellation ordering:
- Normalization method:
- Minimum distance:
- Samples per symbol:

At the bottom are four buttons: OK, Cancel, Help, and Apply.

M-ary number

The number of points in the signal constellation. It must be an even integer.

Input type

Indicates whether the input consists of integers or groups of bits. If this parameter is set to **Bit**, then the **M-ary number** parameter must be 2^K for some positive integer K.

Constellation ordering

Determines how the block maps each group of input bits to a corresponding integer. This field is active only when **Input type** is set to **Bit**.

M-PAM Modulator Baseband

Normalization method

Determines how the block scales the signal constellation. Choices are **Min. distance between symbols**, **Average Power**, and **Peak Power**.

Minimum distance

The distance between two nearest constellation points. This field appears only when **Normalization method** is set to **Min. distance between symbols**.

Average power (watts)

The average power of the symbols in the constellation. This field appears only when **Normalization method** is set to **Average Power**.

Peak power (watts)

The maximum power of the symbols in the constellation. This field appears only when **Normalization method** is set to **Peak Power**.

Samples per symbol

The number of output samples that the block produces for each integer or binary word in the input.

Pair Block M-PAM Demodulator Baseband

See Also General QAM Modulator Baseband

Purpose Modulate using M-ary pulse amplitude modulation

Library AM, in Digital Passband sublibrary of Modulation

Description



The M-PAM Modulator Passband block modulates using M-ary pulse amplitude modulation. The output is a passband representation of the modulated signal. The **M-ary number** parameter, M, is the number of points in the signal constellation. It must be an even integer.

This block uses the baseband equivalent block, M-PAM Modulator Baseband, for internal computations and converts the resulting baseband signal to a passband representation. The following parameters in this block are the same as those of the baseband equivalent block:

- **M-ary number**
- **Input type**
- **Constellation ordering**
- **Normalization method**
- **Minimum distance**
- **Average power**
- **Peak power**

The input must be sample-based. If the **Input type** parameter is **Bit**, then the input must be a vector of length $\log_2(M)$. If the **Input type** parameter is **Integer**, then the input must be a scalar.

Parameters Specific to Passband Simulation

Passband simulation uses a carrier signal. The **Carrier frequency** and **Carrier initial phase** parameters specify the frequency and initial phase, respectively, of the carrier signal. The **Symbol period** parameter must equal the sample time of the input signal, while the **Output sample time** parameter determines the sample time of the output signal.

This block uses a baseband representation of the modulated signal as an intermediate result during internal computations. The **Baseband samples per symbol** parameter indicates how many baseband samples correspond to each integer or binary word in the input, before the block converts them to a passband output.

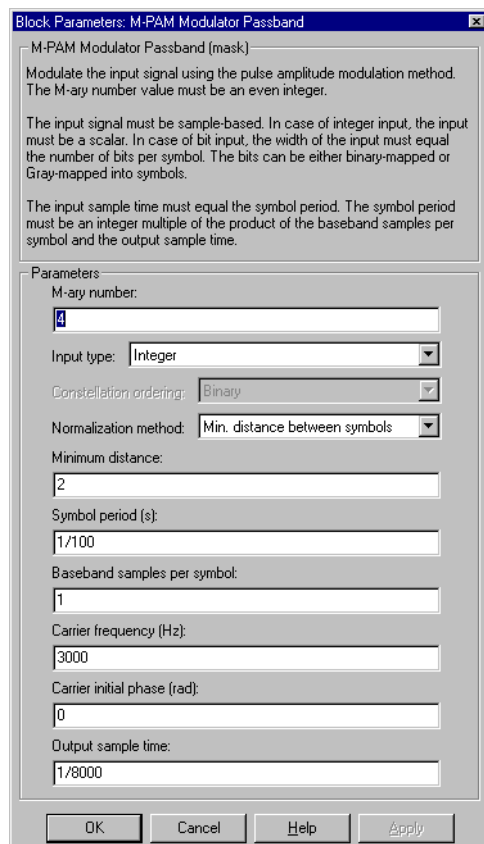
M-PAM Modulator Passband

The timing-related parameters must satisfy these relationships:

- **Symbol period** > (**Carrier frequency**)⁻¹
- **Output sample time** < [2***Carrier frequency** + 2/(**Symbol period**)]⁻¹
- **Symbol period** = K***Output sample time*****Baseband samples per symbol**
for some integer K

Furthermore, **Carrier frequency** is typically much larger than the highest frequency of the unmodulated signal.

Dialog Box



Block Parameters: M-PAM Modulator Passband

M-PAM Modulator Passband (mask)

Modulate the input signal using the pulse amplitude modulation method. The M-ary number value must be an even integer.

The input signal must be sample-based. In case of integer input, the input must be a scalar. In case of bit input, the width of the input must equal the number of bits per symbol. The bits can be either binary-mapped or Gray-mapped into symbols.

The input sample time must equal the symbol period. The symbol period must be an integer multiple of the product of the baseband samples per symbol and the output sample time.

Parameters

M-ary number:

Input type:

Constellation ordering:

Normalization method:

Minimum distance:

Symbol period (s):

Baseband samples per symbol:

Carrier frequency (Hz):

Carrier initial phase (rad):

Output sample time:

OK Cancel Help Apply

M-ary number

The number of points in the signal constellation. It must be an even integer.

Input type

Indicates whether the input consists of integers or groups of bits. If this parameter is set to **Bit**, then the **M-ary number** parameter must be 2^K for some positive integer K .

Constellation ordering

Determines how the block maps each group of input bits to a corresponding integer. This field is active only when **Input type** is set to **Bit**.

Normalization method

Determines how the block scales the signal constellation. Choices are **Min. distance between symbols**, **Average Power**, and **Peak Power**.

Minimum distance

The distance between two nearest constellation points. This field appears only when **Normalization method** is set to **Min. distance between symbols**.

Average power (watts)

The average power of the symbols in the constellation. This field appears only when **Normalization method** is set to **Average Power**.

Peak power (watts)

The maximum power of the symbols in the constellation. This field appears only when **Normalization method** is set to **Peak Power**.

Symbol period (s)

The symbol period, which must equal the sample time of the input.

Baseband samples per symbol

The number of baseband samples that correspond to each integer or binary word in the input, before the block converts them to a passband output.

Carrier frequency (Hz)

The frequency of the carrier.

M-PAM Modulator Passband

- Carrier initial phase (rad)**
The initial phase of the carrier in radians.
- Output sample time**
The sample time of the output signal.

Pair Block M-PAM Demodulator Passband

See Also M-PAM Modulator Baseband

Purpose Demodulate PSK-modulated data

Library PM, in Digital Baseband sublibrary of Modulation

Description



The M-PSK Demodulator Baseband block demodulates a signal that was modulated using the M-ary phase shift keying method. The input is a baseband representation of the modulated signal. The input and output for this block are discrete-time signals. The input can be either a scalar or a frame-based column vector. The **M-ary number** parameter, M, is the number of points in the signal constellation.

Binary or Integer Outputs

If the **Output type** parameter is set to **Integer**, then the block maps the point

$$\exp(j\theta + j2\pi m/M)$$

to m, where θ is the **Phase offset** parameter and m is an integer between 0 and M-1.

If the **Output type** parameter is set to **Bit** and the **M-ary number** parameter has the form 2^K for some positive integer K, then the block outputs binary representations of integers between 0 and M-1. It outputs a group of K bits, called a binary *word*, for each symbol.

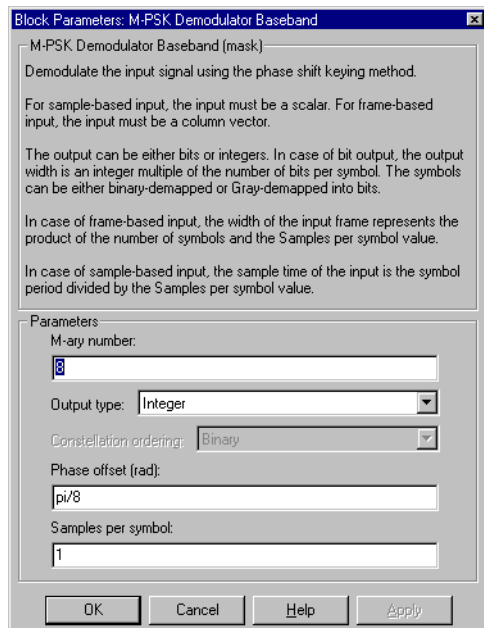
In binary output mode, the **Constellation ordering** parameter indicates how the block maps an integer to a corresponding group of K output bits. See the reference page for the M-PSK Modulator Baseband block for details.

Processing an Upsampled Modulated Signal

The input signal can be an upsampled version of the modulated signal. The **Samples per symbol** parameter is the upsampling factor. It must be a positive integer. For more information, see “Upsampled Signals and Rate Changes” on page 2-72.

M-PSK Demodulator Baseband

Dialog Box



M-ary number

The number of points in the signal constellation.

Output type

Determines whether the output consists of integers or groups of bits. If this parameter is set to **Bit**, then the **M-ary number** parameter must be 2^K for some positive integer K .

Constellation ordering

Determines how the block maps each integer to a group of output bits. This field is active only when **Output type** is set to **Bit**.

Phase offset (rad)

The phase of the zeroth point of the signal constellation.

Samples per symbol

The number of input samples that represent each modulated symbol.

Pair Block M-PSK Modulator Baseband

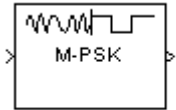
See Also BPSK Demodulator Baseband, QPSK Demodulator Baseband, M-DPSK
Demodulator Baseband

M-PSK Demodulator Passband

Purpose Demodulate PSK-modulated data

Library PM, in Digital Passband sublibrary of Modulation

Description



The M-PSK Demodulator Passband block demodulates a signal that was modulated using the M-ary phase shift keying method. The input is a passband representation of the modulated signal. The **M-ary number** parameter, M, is the number of points in the signal constellation.

This block converts the input to an equivalent baseband representation and then uses the baseband equivalent block, M-PSK Demodulator Baseband, for internal computations. The following parameters in this block are the same as those of the baseband equivalent block:

- **M-ary number**
- **Output type**
- **Constellation ordering**

The input must be a sample-based scalar signal.

Parameters Specific to Passband Simulation

Passband simulation uses a carrier signal. The **Carrier frequency** and **Carrier initial phase** parameters specify the frequency and initial phase, respectively, of the carrier signal. The **Input sample time** parameter specifies the sample time of the input signal, while the **Symbol period** parameter equals the sample time of the output signal.

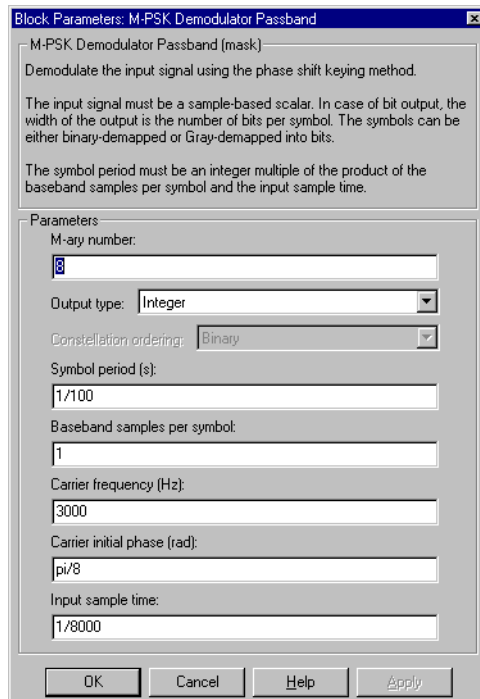
This block uses a baseband representation of the modulated signal as an intermediate signal during internal computations. The **Baseband samples per symbol** parameter indicates how many baseband samples correspond to each integer or binary word in the output.

The timing-related parameters must satisfy these relationships:

- **Input sample time** > $(\text{Carrier frequency})^{-1}$
- **Symbol period** < $[2 * \text{Carrier frequency} + 2 / (\text{Input sample time})]^{-1}$
- **Input sample time** = $K * \text{Symbol period} * \text{Baseband samples per symbol}$ for some integer K

Also, this block incurs an extra output period of delay compared to its baseband equivalent block.

Dialog Box



The dialog box is titled "Block Parameters: M-PSK Demodulator Passband". It contains the following text and controls:

M-PSK Demodulator Passband (mask)
Demodulate the input signal using the phase shift keying method.

The input signal must be a sample-based scalar. In case of bit output, the width of the output is the number of bits per symbol. The symbols can be either binary-demapped or Gray-demapped into bits.

The symbol period must be an integer multiple of the product of the baseband samples per symbol and the input sample time.

Parameters

M-ary number:

Output type:

Constellation ordering:

Symbol period (s):

Baseband samples per symbol:

Carrier frequency (Hz):

Carrier initial phase (rad):

Input sample time:

Buttons: OK, Cancel, Help, Apply

M-ary number

The number of points in the signal constellation.

Output type

Determines whether the output consists of integers or groups of bits. If this parameter is set to **Bit**, then the **M-ary number** parameter must be 2^K for some positive integer K.

Constellation ordering

Determines how the block maps each integer to a group of output bits. This field is active only when **Output type** is set to **Bit**.

M-PSK Demodulator Passband

Symbol period (s)

The symbol period, which equals the sample time of the output.

Baseband samples per symbol

The number of baseband samples that represent each modulated symbol, after the block converts the passband input to a baseband intermediary signal.

Carrier frequency (Hz)

The frequency of the carrier.

Carrier initial phase (rad)

The initial phase of the carrier in radians.

Input sample time

The sample time of the input signal.

Pair Block	M-PSK Modulator Passband
See Also	M-PSK Demodulator Baseband

Purpose Modulate using the M-ary phase shift keying method

Library PM, in Digital Baseband sublibrary of Modulation

Description



The M-PSK Modulator Baseband block modulates using the M-ary phase shift keying method. The output is a baseband representation of the modulated signal. The **M-ary number** parameter, M, is the number of points in the signal constellation.

Baseband M-ary phase shift keying modulation with a phase offset of θ maps an integer m between 0 and $M-1$ to the complex value

$$\exp(j\theta + j2\pi m/M)$$

The input and output for this block are discrete-time signals. To use integers between 0 and $M-1$ as input values, set the **Input type** parameter to **Integer**. In this case, the input can be either a scalar or a frame-based column vector.

Alternative configurations of the block determine how the block interprets its input and arranges its output, as explained in the sections below.

Binary Inputs

If the **Input type** parameter is set to **Bit** and the **M-ary number** parameter has the form 2^K for some positive integer K , then the block accepts binary representations of integers between 0 and $M-1$. It modulates each group of K bits, called a binary *word*. The input can be either a vector of length K or a frame-based column vector whose length is an integer multiple of K .

In binary input mode, the **Constellation ordering** parameter indicates how the block maps a group of K input bits to a corresponding integer. Choices are **Binary** and **Gray**. For more information, see “Binary-Valued and Integer-Valued Signals” on page 2-68.

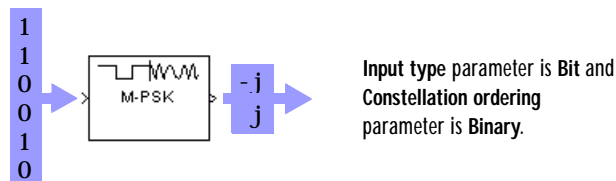
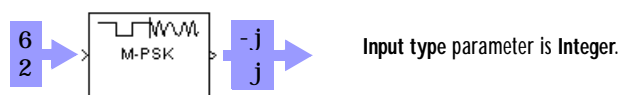
If **Constellation ordering** is set to **Gray**, then the block uses a Gray-coded signal constellation; as a result, binary representations that differ in more than one bit cannot map to consecutive integers modulo M . The explicit mapping is described in “Algorithm” below.

M-PSK Modulator Baseband

Frame-Based Inputs

If the input is a frame-based column vector, then the block processes several integers or several binary words, in each time step. (If the **Input type** parameter is set to **Bit**, then a binary word consists of $\log_2(M)$ bits.)

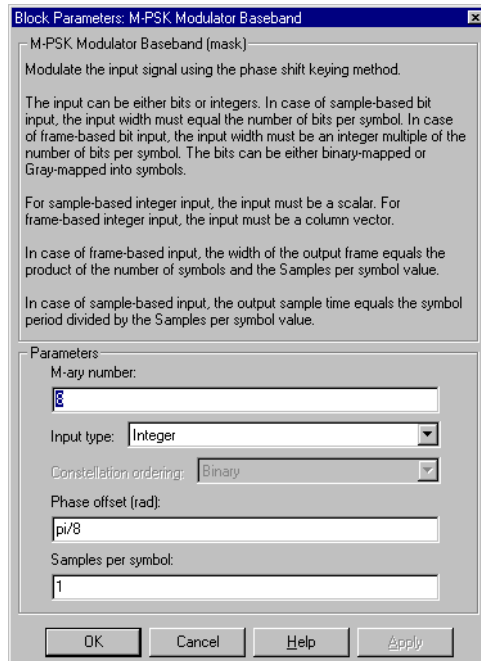
For example, the schematics below illustrate how the block processes two 8-ary integers or binary words in one time step. The signals involved are all frame-based column vectors. In both cases, the **Phase offset** parameter is 0.



Upsampling the Modulated Signal

This block can output an upsampled version of the modulated signal. The **Samples per symbol** parameter is the upsampling factor. It must be a positive integer. For more information, see “Upsampled Signals and Rate Changes” on page 2-72.

Dialog Box



M-ary number

The number of points in the signal constellation.

Input type

Indicates whether the input consists of integers or groups of bits. If this parameter is set to **Bit**, then the **M-ary number** parameter must be 2^K for some positive integer K .

Constellation ordering

Determines how the block maps each group of input bits to a corresponding integer. This field is active only when **Input type** is set to **Bit**.

Phase offset (rad)

The phase of the zeroth point of the signal constellation.

Samples per symbol

The number of output samples that the block produces for each integer or binary word in the input.

Algorithm

If the **Constellation ordering** parameter is set to **Gray**, then the block internally assigns the binary inputs to points of a predefined Gray-coded signal constellation. The block's predefined M-ary Gray-coded signal constellation assigns the binary representation

$$\text{de2bi}(\text{bitxor}(m, \text{floor}(m/2)), \log_2(M), \text{'left-msb'})$$

to the m th phase. The zeroth phase in the constellation is the **Phase offset** parameter, and successive phases are counted in a counterclockwise direction.

Note This transformation might seem counterintuitive because it constitutes a Gray-to-binary mapping. However, the block must use it to impose a Gray ordering on the signal constellation, which has a natural binary ordering.

In other words, if the block input is the natural binary representation, u , of the integer U , then the block output has phase

$$j\theta + j2\pi m/M$$

where θ is the **Phase offset** parameter and m is an integer between 0 and $M-1$ that satisfies

$$m \text{ XOR } \lfloor m/2 \rfloor = U$$

For example, if $M = 8$, then the binary representations that correspond to the zeroth through seventh phases are below.

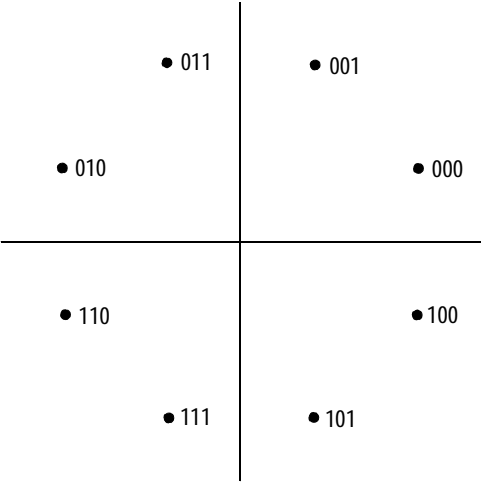
$$M = 8; \quad m = [0:M-1]'; \\ \text{de2bi}(\text{bitxor}(m, \text{floor}(m/2)), \log_2(M), \text{'left-msb'})$$

ans =

0	0	0
0	0	1
0	1	1
0	1	0

1	1	0
1	1	1
1	0	1
1	0	0

Below is the 8-ary Gray-coded constellation that the block uses if the **Phase offset** parameter is $\pi/8$.



Pair Block

M-PSK Demodulator Baseband

See Also

BPSK Modulator Baseband, QPSK Modulator Baseband, M-DPSK Modulator Baseband

M-PSK Modulator Passband

Purpose Modulate using the M-ary phase shift keying method

Library PM, in Digital Passband sublibrary of Modulation

Description



The M-PSK Modulator Passband block modulates using the M-ary phase shift keying method. The output is a passband representation of the modulated signal. The **M-ary number** parameter, M, is the number of points in the signal constellation.

This block uses the baseband equivalent block, M-PSK Modulator Baseband, for internal computations and converts the resulting baseband signal to a passband representation. The following parameters in this block are the same as those of the baseband equivalent block:

- **M-ary number**
- **Input type**
- **Constellation ordering**

The input must be sample-based. If the **Input type** parameter is **Bit**, then the input must be a vector of length $\log_2(M)$. If the **Input type** parameter is **Integer**, then the input must be a scalar.

Parameters Specific to Passband Simulation

Passband simulation uses a carrier signal. The **Carrier frequency** and **Carrier initial phase** parameters specify the frequency and initial phase, respectively, of the carrier signal. The **Symbol period** parameter must equal the sample time of the input signal, while the **Output sample time** parameter determines the sample time of the output signal.

This block uses a baseband representation of the modulated signal as an intermediate result during internal computations. The **Baseband samples per symbol** parameter indicates how many baseband samples correspond to each integer or binary word in the input, before the block converts them to a passband output.

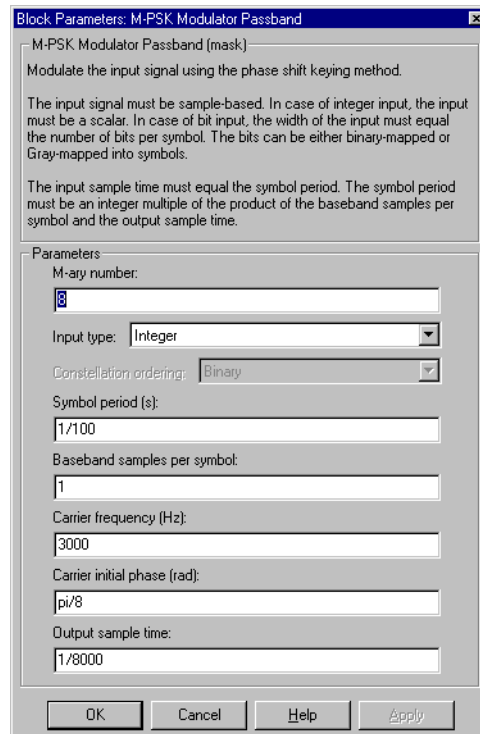
The timing-related parameters must satisfy these relationships:

- **Symbol period** > (**Carrier frequency**)⁻¹
- **Output sample time** < [2***Carrier frequency** + 2/(**Symbol period**)]⁻¹

- **Symbol period** = $K \times \text{Output sample time} \times \text{Baseband samples per symbol}$ for some integer K

Furthermore, **Carrier frequency** is typically much larger than the highest frequency of the unmodulated signal.

Dialog Box



Block Parameters: M-PSK Modulator Passband

M-PSK Modulator Passband (mask)

Modulate the input signal using the phase shift keying method.

The input signal must be sample-based. In case of integer input, the input must be a scalar. In case of bit input, the width of the input must equal the number of bits per symbol. The bits can be either binary-mapped or Gray-mapped into symbols.

The input sample time must equal the symbol period. The symbol period must be an integer multiple of the product of the baseband samples per symbol and the output sample time.

Parameters

M-ary number:

Input type:

Constellation ordering:

Symbol period (s):

Baseband samples per symbol:

Carrier frequency (Hz):

Carrier initial phase (rad):

Output sample time:

OK Cancel Help Apply

M-ary number

The number of points in the signal constellation.

Input type

Indicates whether the input consists of integers or groups of bits. If this parameter is set to **Bit**, then the **M-ary number** parameter must be 2^K for some positive integer K .

M-PSK Modulator Passband

Constellation ordering

Determines how the block maps each group of input bits to a corresponding integer. This field is active only when **Input type** is set to **Bit**.

Symbol period (s)

The symbol period, which must equal the sample time of the input.

Baseband samples per symbol

The number of baseband samples that correspond to each integer or binary word in the input, before the block converts them to a passband output.

Carrier frequency (Hz)

The frequency of the carrier.

Carrier initial phase (rad)

The initial phase of the carrier in radians.

Output sample time

The sample time of the output signal.

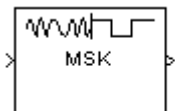
Pair Block M-PSK Demodulator Passband

See Also M-PSK Modulator Baseband

Purpose Demodulate MSK-modulated data

Library CPM, in Digital Baseband sublibrary of Modulation

Description



The MSK Demodulator Baseband block demodulates a signal that was modulated using the minimum shift keying method. The input is a baseband representation of the modulated signal. The **Phase offset** parameter is the initial phase of the modulated waveform.

Traceback Length and Output Delays

Internally, this block creates a trellis description of the modulation scheme and uses the Viterbi algorithm. The **Traceback length** parameter, D , in this block is the number of trellis branches used to construct each traceback path. D influences the output delay, which is the number of zero symbols that precede the first meaningful demodulated value in the output.

- If the input signal is sample-based, then the delay consists of $D+1$ zero symbols.
- If the input signal is frame-based, then the delay consists of D zero symbols.

Inputs and Outputs

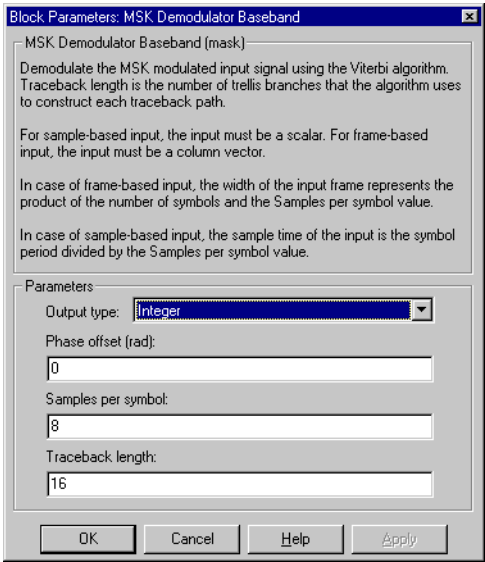
The input can be either a scalar or a frame-based column vector. If the **Output type** parameter is set to **Integer**, then the block produces values of 1 and -1. If the **Output type** parameter is set to **Bit**, then the block produces values of 0 and 1.

Processing an Upsampled Modulated Signal

The input signal can be an upsampled version of the modulated signal. The **Samples per symbol** parameter is the upsampling factor. It must be a positive integer. For more information, see “Upsampled Signals and Rate Changes” on page 2-72.

MSK Demodulator Baseband

Dialog Box



Output type

Determines whether the output consists of bipolar or binary values.

Phase offset (rad)

The initial phase of the modulated waveform.

Samples per symbol

The number of input samples that represent each modulated symbol.

Traceback length

The number of trellis branches that the Viterbi Decoder block uses to construct each traceback path.

Pair Block

MSK Modulator Baseband

See Also

CPM Demodulator Baseband, Viterbi Decoder

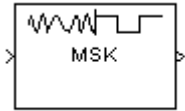
References

[1] Anderson, John B., Tor Aulin, and Carl-Erik Sundberg. *Digital Phase Modulation*. New York: Plenum Press, 1986.

Purpose Demodulate MSK-modulated data

Library CPM, in Digital Passband sublibrary of Modulation

Description



The MSK Demodulator Passband block demodulates a signal that was modulated using the minimum shift keying method. The input is a passband representation of the modulated signal.

This block converts the input to an equivalent baseband representation and then uses the baseband equivalent block, MSK Demodulator Baseband, for internal computations. The following parameters in this block are the same as those of the baseband equivalent block:

- **Output type**
- **Traceback length**

The input must be a sample-based scalar signal.

Parameters Specific to Passband Simulation

Passband simulation uses a carrier signal. The **Carrier frequency** and **Carrier initial phase** parameters specify the frequency and initial phase, respectively, of the carrier signal. The **Input sample time** parameter specifies the sample time of the input signal, while the **Symbol period** parameter equals the sample time of the output signal.

This block uses a baseband representation of the modulated signal as an intermediate signal during internal computations. The **Baseband samples per symbol** parameter indicates how many baseband samples correspond to each integer or binary word in the output.

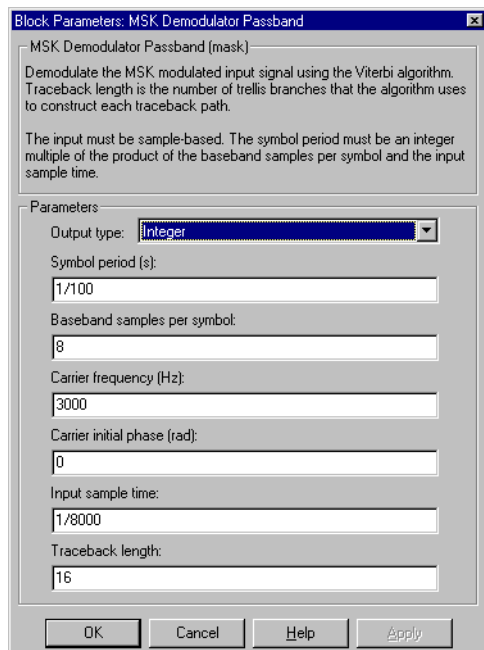
The timing-related parameters must satisfy these relationships:

- **Input sample time** > (**Carrier frequency**)⁻¹
- **Symbol period** < [2***Carrier frequency** + 2/(**Input sample time**)]⁻¹
- **Input sample time** = K***Symbol period*****Baseband samples per symbol** for some integer K

Also, this block incurs an extra output period of delay compared to its baseband equivalent block.

MSK Demodulator Passband

Dialog Box



Output type

Determines whether the output consists of bipolar or binary values.

Symbol period (s)

The symbol period, which equals the sample time of the output.

Baseband samples per symbol

The number of baseband samples that represent each modulated symbol, after the block converts the passband input to a baseband intermediary signal.

Carrier frequency (Hz)

The frequency of the carrier.

Carrier initial phase (rad)

The initial phase of the carrier in radians.

Input sample time

The sample time of the input signal.

Traceback length

The number of trellis branches that the Viterbi Decoder block uses to construct each traceback path.

Pair Block MSK Modulator Passband

See Also MSK Demodulator Baseband, Viterbi Decoder

References [1] Anderson, John B., Tor Aulin, and Carl-Erik Sundberg. *Digital Phase Modulation*. New York: Plenum Press, 1986.

MSK Modulator Baseband

Purpose

Modulate using the minimum shift keying method

Library

CPM, in Digital Baseband sublibrary of Modulation

Description



The MSK Modulator Baseband block modulates using the minimum shift keying method. The output is a baseband representation of the modulated signal.

The **Modulation index** parameter times π radians is the phase shift due to the latest symbol when that symbol is the integer 1. The **Phase offset** parameter is the initial phase of the output waveform, measured in radians.

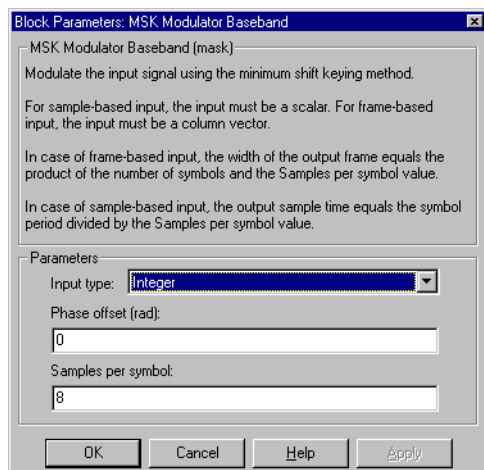
Input Attributes

The input can be either a scalar or a frame-based column vector. If the **Input type** parameter is set to **Integer**, then the block accepts values of 1 and -1. If the **Input type** parameter is set to **Bit**, then the block accepts values of 0 and 1.

Upsampling the Modulated Signal

This block can output an upsampled version of the modulated signal. The **Samples per symbol** parameter is the upsampling factor. It must be a positive integer. For more information, see “Upsampled Signals and Rate Changes” on page 2-72.

Dialog Box



Input type

Indicates whether the input consists of bipolar or binary values.

Phase offset (rad)

The initial phase of the output waveform.

Samples per symbol

The number of output samples that the block produces for each integer or bit in the input.

Pair Block MSK Demodulator Baseband

See Also CPM Modulator Baseband

References [1] Anderson, John B., Tor Aulin, and Carl-Erik Sundberg. *Digital Phase Modulation*. New York: Plenum Press, 1986.

MSK Modulator Passband

Purpose Modulate using the minimum shift keying method

Library CPM, in Digital Passband sublibrary of Modulation

Description



The MSK Modulator Passband block modulates using the minimum shift keying method. The output is a passband representation of the modulated signal.

The input must be sample-based. If the **Input type** parameter is **Bit**, then the input must be a vector of length $\log_2(M)$, containing values of 0 and 1. If the **Input type** parameter is **Integer**, then the input must be a scalar containing values of 1 and -1.

Parameters Specific to Passband Simulation

Passband simulation uses a carrier signal. The **Carrier frequency** and **Carrier initial phase** parameters specify the frequency and initial phase, respectively, of the carrier signal. The **Symbol period** parameter must equal the sample time of the input signal, while the **Output sample time** parameter determines the sample time of the output signal.

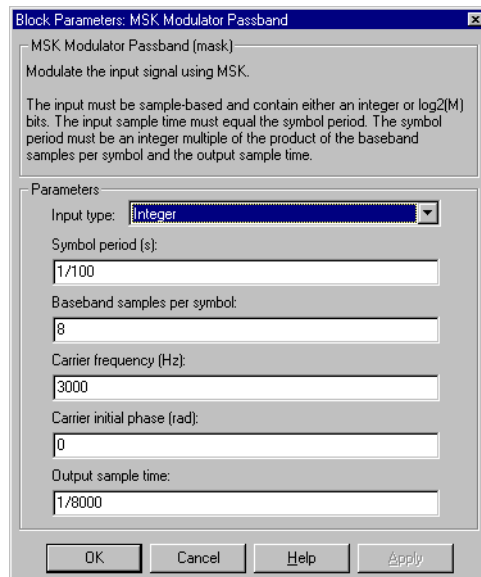
This block uses a baseband representation of the modulated signal as an intermediate result during internal computations. The **Baseband samples per symbol** parameter indicates how many baseband samples correspond to each integer or binary word in the input, before the block converts them to a passband output.

The timing-related parameters must satisfy these relationships:

- **Symbol period** > (**Carrier frequency**)⁻¹
- **Output sample time** < [2***Carrier frequency** + 2/(**Symbol period**)]⁻¹
- **Symbol period** = K***Output sample time*****Baseband samples per symbol**
for some integer K

Furthermore, **Carrier frequency** is typically much larger than the highest frequency of the unmodulated signal.

Dialog Box



Input type

Indicates whether the input consists of bipolar or binary values.

Symbol period (s)

The symbol period, which must equal the sample time of the input.

Baseband samples per symbol

The number of baseband samples that correspond to each integer or binary word in the input, before the block converts them to a passband output.

Carrier frequency (Hz)

The frequency of the carrier.

Carrier initial phase (rad)

The initial phase of the carrier in radians.

Output sample time

The sample time of the output signal.

Pair Block

MSK Demodulator Passband

MSK Modulator Passband

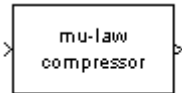
See Also MSK Modulator Baseband

References [1] Anderson, John B., Tor Aulin, and Carl-Erik Sundberg. *Digital Phase Modulation*. New York: Plenum Press, 1986.

Purpose Implement μ -law compressor for source coding

Library Source Coding

Description The Mu-Law Compressor block implements a μ -law compressor for the input signal. The formula for the μ -law compressor is

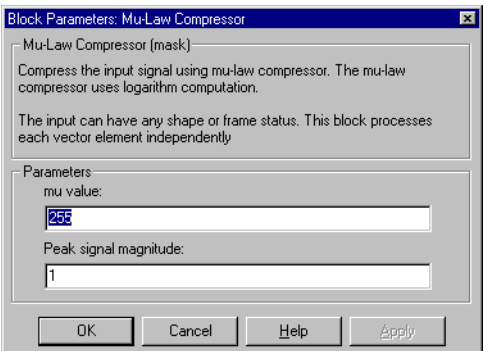


$$y = \frac{V \log(1 + \mu |x| / V)}{\log(1 + \mu)} \operatorname{sgn}(x)$$

where μ is the μ -law parameter of the compressor, V is the peak magnitude of x , \log is the natural logarithm, and sgn is the signum function (sign in MATLAB).

The input can have any shape or frame status. This block processes each vector element independently.

Dialog Box



mu value

The μ -law parameter of the compressor.

Peak signal magnitude

The peak value of the input signal. This is also the peak value of the output.

Pair Block Mu-Law Expander

See Also A-Law Compressor

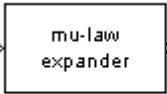
References [1] Sklar, Bernard. *Digital Communications: Fundamentals and Applications*. Englewood Cliffs, N.J.: Prentice-Hall, 1988.

Mu-Law Expander

Purpose Implement μ -law expander for source coding

Library Source Coding

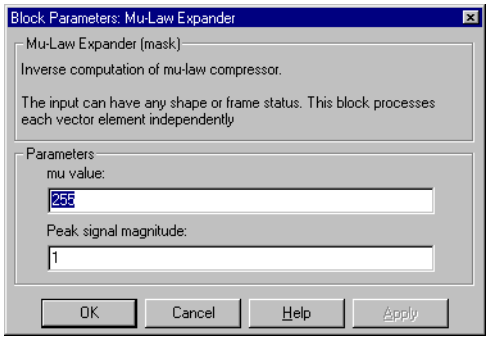
Description The Mu-Law Expander block recovers data that the Mu-Law Compressor block compressed. The formula for the μ -law expander, shown below, is the inverse of the compressor function.



$$x = \frac{V}{\mu} (e^{|y| \log(1 + \mu)/V} - 1) \text{sgn}(y)$$

The input can have any shape or frame status. This block processes each vector element independently.

Dialog Box



mu value

The μ -law parameter of the compressor.

Peak signal magnitude

The peak value of the input signal. This is also the peak value of the output.

Pair Block Mu-Law Compressor

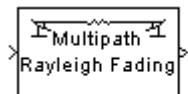
See Also A-Law Expander

References [1] Sklar, Bernard. *Digital Communications: Fundamentals and Applications*. Englewood Cliffs, N.J.: Prentice-Hall, 1988.

Purpose Simulate a multipath Rayleigh fading propagation channel

Library Channels

Description



The Multipath Rayleigh Fading Channel block implements a baseband simulation of a multipath Rayleigh fading propagation channel. This block is useful for modeling mobile wireless communication systems. For details about fading channels, see the works listed in “References” on page 4-333.

The input can be either a scalar or a frame-based column vector. The input is a complex signal.

Relative motion between the transmitter and receiver causes Doppler shifts in the signal frequency. The **Doppler frequency** parameter is the *maximum* Doppler shift that the signal undergoes. The Jakes PSD (power spectral density) determines the spectrum of the Rayleigh process.

Since a multipath channel reflects signals at multiple places, a transmitted signal travels to the receiver along several paths that may have different lengths and hence different associated time delays. Fading occurs when signals traveling along different paths interfere with each other. In the block’s parameter mask, the **Delay vector** specifies the time delay for each path. If the **Normalize gain vector to 0 dB overall gain** box is unchecked, then the **Gain vector** specifies the gain for each path. If the box is checked, then the block uses a multiple of **Gain vector** instead of the **Gain vector** itself, choosing the scaling factor so that the channel’s effective gain considering all paths is 0 dB.

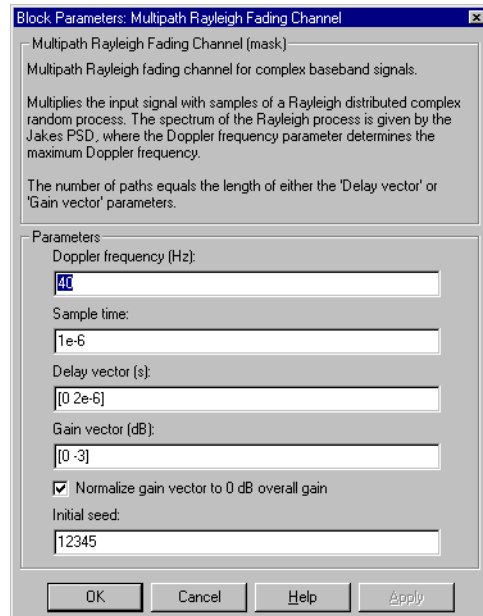
The number of paths is the length of **Delay vector** or **Gain vector**, whichever is larger. If both of these parameters are vectors, then they must have the same length; if exactly one of these parameters is a scalar, then the block expands it into a vector whose size matches that of the other **vector** parameter.

The **Sample time** parameter is the time between successive elements of the input signal. Note that if the input is a frame-based column vector of length n , then the frame period (as Simulink’s Probe block reports, for example) is $n \times \text{Sample time}$.

The block multiplies the input signal by samples of a Rayleigh-distributed complex random process. The scalar **Initial seed** parameter seeds the random number generator.

Multipath Rayleigh Fading Channel

Dialog Box



Doppler frequency (Hz)

A positive scalar that indicates the maximum Doppler shift.

Sample time

The period of each element of the input signal.

Delay vector (s)

A vector that specifies the propagation delay for each path.

Gain vector (dB)

A vector that specifies the gain for each path.

Normalize gain vector to 0 dB overall gain

Checking this box causes the block to scale the **Gain vector** parameter so that the channel's effective gain (considering all paths) is 0 decibels.

Initial seed

The scalar seed for the Gaussian noise generator.

Algorithm

This implementation is based on the direct form simulator described in [1].

Some wireless applications, such as standard GSM (Global System for Mobile Communication) systems, prefer to specify Doppler shifts in terms of the speed of the mobile. If the mobile moves at speed v making an angle of θ with the direction of wave motion, then the Doppler shift is

$$f_d = (vf/c) \cos \theta$$

where f is the transmission carrier frequency and c is the speed of light. The Doppler frequency is the maximum Doppler shift arising from motion of the mobile.

See Also

Rayleigh Noise Generator, Rician Fading Channel

References

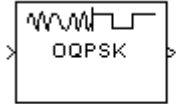
- [1] Fechtel, Stefan A. "A Novel Approach to Modeling and Efficient Simulation of Frequency-Selective Fading Radio Channels." *IEEE Journal on Selected Areas in Communications*, vol. 11, April 1993. 422-431.
- [2] Jakes, William C., ed. *Microwave Mobile Communications*. New York: IEEE Press, 1974.
- [3] Lee, William C. Y. *Mobile Communications Design Fundamentals*, 2nd ed. New York: Wiley, 1993.

OQPSK Demodulator Baseband

Purpose Demodulate OQPSK-modulated data

Library PM, in Digital Baseband sublibrary of Modulation

Description



The OQPSK Demodulator Baseband block demodulates a signal that was modulated using the offset quadrature phase shift keying method. The input is a baseband representation of the modulated signal.

The input must be a discrete-time complex signal. The input can be either a scalar or a frame-based column vector.

If the **Output type** parameter is set to **Integer**, then the block outputs integers between 0 and 3. If the **Output type** parameter is set to **Bit**, then the block outputs binary representations of such integers, in a binary-valued vector whose length is an even number.

The input symbol period is half the period of each output integer or bit pair. The constellation used to map bit pairs to symbols is on the reference page for the OQPSK Modulator Baseband block.

Frame-Based Inputs

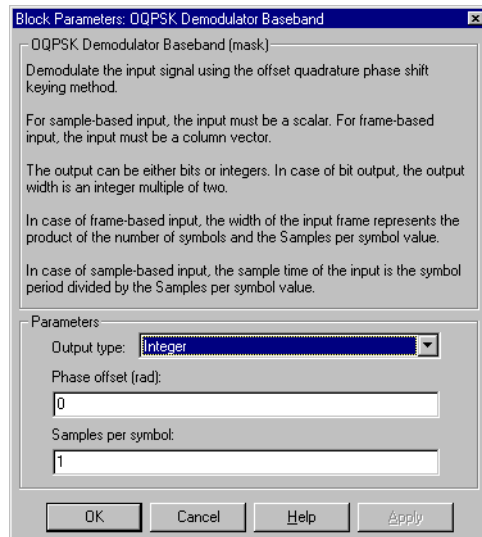
If the input is a frame-based column vector, then the block processes several integers or several pairs of bits, in each time step. In this case, the output sample time equals the input sample time, even though the symbol period is half the output period.

Processing an Upsampled Modulated Signal

The input signal can be an upsampled version of the modulated signal. The **Samples per symbol** parameter is the upsampling factor. It must be a positive integer.

- If the input is a frame-based column vector, then the output vector contains $1/2S$ integers or pairs of bits for each sample in the input vector, while the output sample time equals the input sample time.
- If the input is a sample-based scalar, then the output vector contains a single integer or pair of bits, while the output sample time is $2S$ times the input sample time.

Dialog Box



Output type

Determines whether the output consists of integers or pairs of bits.

Phase offset (rad)

The amount by which the phase of the zeroth point of the signal constellation is shifted from $\pi/4$.

Samples per symbol

The number of input samples that represent each modulated symbol.

Pair Block

OQPSK Modulator Baseband

See Also

QPSK Demodulator Baseband

OQPSK Demodulator Passband

Purpose Demodulate OQPSK-modulated data

Library PM, in Digital Passband sublibrary of Modulation

Description



The OQPSK Demodulator Passband block demodulates a signal that was modulated using the offset quadrature phase shift keying method. The input is a passband representation of the modulated signal.

If the **Output type** parameter is set to **Integer**, then the block outputs integers between 0 and 3. If the **Output type** parameter is set to **Bit**, then the block outputs binary representations of such integers, in binary-valued vectors of length two. The constellation used to map bit pairs to symbols is on the reference page for the OQPSK Modulator Passband block.

The input must be a sample-based scalar signal.

Parameters Specific to Passband Simulation

Passband simulation uses a carrier signal. The **Carrier frequency** and **Carrier initial phase** parameters specify the frequency and initial phase, respectively, of the carrier signal. The **Input sample time** parameter specifies the sample time of the input signal, while the **Symbol period** parameter equals the sample time of the output signal.

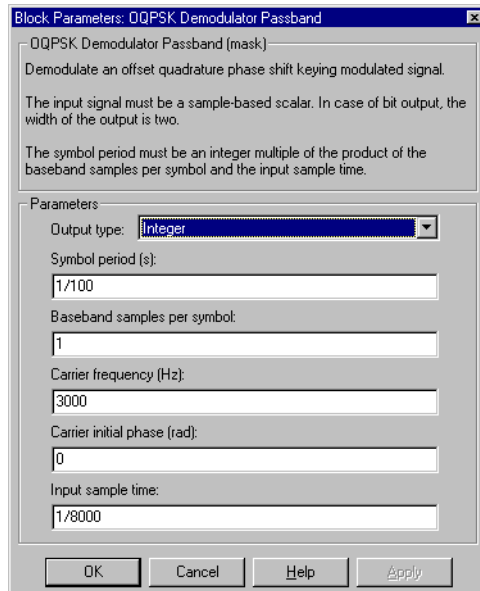
This block uses a baseband representation of the modulated signal as an intermediate signal during internal computations. The **Baseband samples per symbol** parameter indicates how many baseband samples correspond to each integer or binary word in the output.

The timing-related parameters must satisfy these relationships:

- **Input sample time** > (**Carrier frequency**)⁻¹
- **Symbol period** < [2***Carrier frequency** + 2/(**Input sample time**)]⁻¹
- **Input sample time** = K***Symbol period*****Baseband samples per symbol** for some integer K

Also, this block incurs an extra output period of delay compared to its baseband equivalent block.

Dialog Box



Output type

Indicates whether the output consists of integers or groups of bits.

Symbol period (s)

The symbol period, which equals the sample time of the output.

Baseband samples per symbol

The number of baseband samples that represent each modulated symbol, after the block converts the passband input to a baseband intermediary signal.

Carrier frequency (Hz)

The frequency of the carrier.

Carrier initial phase (rad)

The initial phase of the carrier in radians.

Input sample time

The sample time of the input signal.

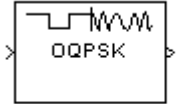
OQPSK Demodulator Passband

Pair Block	OQPSK Modulator Passband
See Also	OQPSK Demodulator Baseband

Purpose Modulate using the offset quadrature phase shift keying method

Library PM, in Digital Baseband sublibrary of Modulation

Description The OQPSK Modulator Baseband block modulates using the offset quadrature phase shift keying method. The output is a baseband representation of the modulated signal.

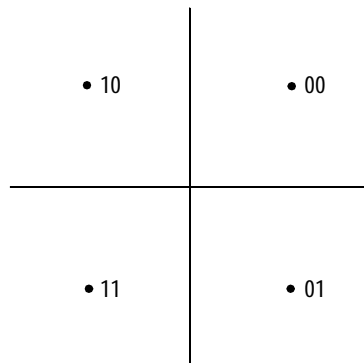


If the **Input type** parameter is set to **Integer**, then valid input values are 0, 1, 2, and 3. In this case, the input can be either a scalar or a frame-based column vector.

If the **Input type** parameter is set to **Bit**, then the input must be a binary-valued vector. In this case, the input can be either a vector of length two or a frame-based column vector whose length is an even integer.

The symbol period is half the input period. The first output symbol is an initial condition of zero that is unrelated to the input values.

The constellation used to map bit pairs to symbols is in the figure below. If the block's **Phase offset** parameter is nonzero, then this constellation is rotated by that parameter value.



Frame-Based Inputs

If the input is a frame-based column vector, then the block processes several integers or several pairs of bits in each time step. In this case, the output

OQPSK Modulator Baseband

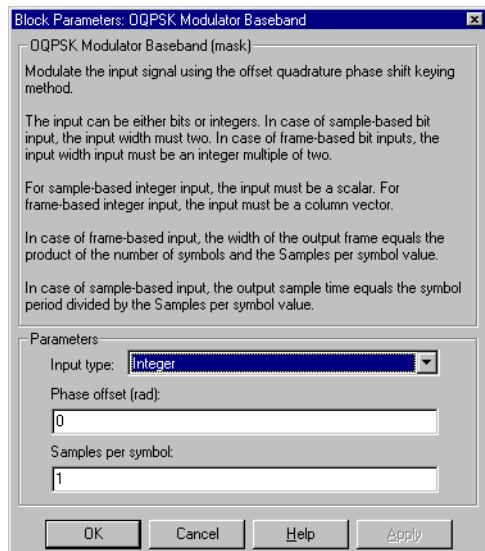
sample time equals the input sample time, even though the period of each output symbol is half the period of each integer or bit pair in the input.

Upsampling the Modulated Signal

This block can output an upsampled version of the modulated signal. The **Samples per symbol** parameter, S , is the upsampling factor. It must be a positive integer.

- If the input is a frame-based column vector, then the output vector length is $2S$ times the number of integers or pairs of bits in the input vector, while the output sample time equals the input sample time. The one-symbol initial condition inherent in this block corresponds to the first S elements of the first output vector.
- If the input is a sample-based scalar, then the output vector is a scalar, while the output sample time is $1/2S$ times the input sample time. The one-symbol initial condition inherent in this block corresponds to the first S samples.

Dialog Box



Input type

Indicates whether the input consists of integers or pairs of bits.

Phase offset (rad)

The amount by which the phase of the zeroth point of the signal constellation is shifted from $\pi/4$.

Samples per symbol

The number of output samples that the block produces for each integer or pair of bits in the input.

Pair Block OQPSK Demodulator Baseband

See Also QPSK Modulator Baseband

OQPSK Modulator Passband

Purpose Modulate using the offset quadrature phase shift keying method

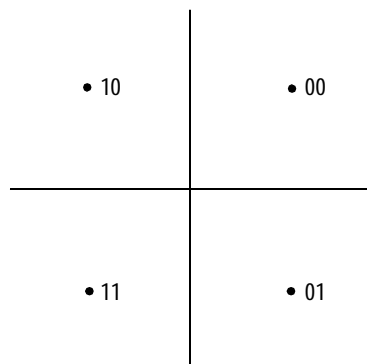
Library PM, in Digital Passband sublibrary of Modulation

Description The OQPSK Modulator Passband block modulates using the offset quadrature phase shift keying method. The output is a passband representation of the modulated signal.



If the **Input type** parameter is set to **Integer**, then valid input values are 0, 1, 2, and 3. In this case, the input must be a sample-based scalar. If the **Input type** parameter is set to **Bit**, then the input must be a binary-valued sample-based vector of length two.

The constellation used to map bit pairs to symbols is in the figure below.



Parameters Specific to Passband Simulation

Passband simulation uses a carrier signal. The **Carrier frequency** and **Carrier initial phase** parameters specify the frequency and initial phase, respectively, of the carrier signal. The **Symbol period** parameter must equal the sample time of the input signal, while the **Output sample time** parameter determines the sample time of the output signal.

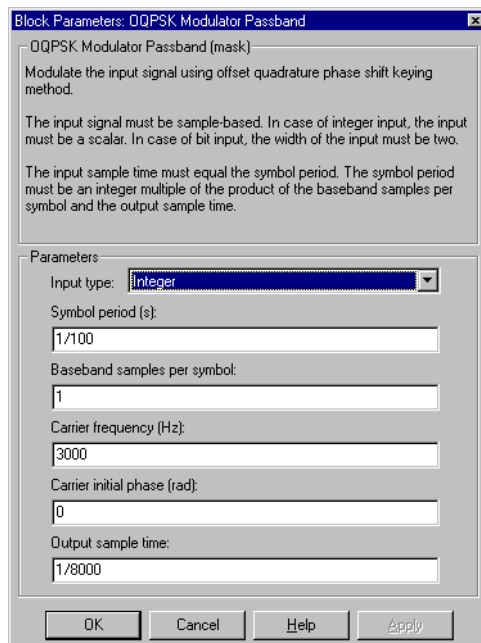
This block uses a baseband representation of the modulated signal as an intermediate result during internal computations. The **Baseband samples per symbol** parameter indicates how many baseband samples correspond to each integer or binary word in the input, before the block converts them to a passband output.

The timing-related parameters must satisfy these relationships:

- **Symbol period** > (**Carrier frequency**)⁻¹
- **Output sample time** < [2***Carrier frequency** + 2/(**Symbol period**)]⁻¹
- **Symbol period** = K***Output sample time*****Baseband samples per symbol**
for some integer K

Furthermore, **Carrier frequency** is typically much larger than the highest frequency of the unmodulated signal.

Dialog Box



Input type

Indicates whether the input consists of integers or pairs of bits. If this parameter is set to **Bit**, then the **M-ary number** parameter must be 2^K for some positive integer K.

Symbol period (s)

The symbol period, which must equal the sample time of the input.

OQPSK Modulator Passband

Baseband samples per symbol

The number of baseband samples that correspond to each integer or pair of bits in the input, before the block converts them to a passband output.

Carrier frequency (Hz)

The frequency of the carrier.

Carrier initial phase (rad)

The initial phase of the carrier in radians.

Output sample time

The sample time of the output signal.

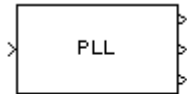
Pair Block OQPSK Demodulator Passband

See Also OQPSK Modulator Baseband

Purpose Implement a phase-locked loop to recover the phase of the input signal

Library Synchronization

Description



The Phase-Locked Loop (PLL) block is a feedback control system that automatically adjusts the phase of a locally generated signal to match the phase of an input signal. This block is most appropriate when the input is a narrowband signal.

This PLL has these three components:

- A multiplier used as a phase detector.
- A filter. You specify the filter's transfer function using the **Lowpass filter numerator** and **Lowpass filter denominator** mask parameters. Each is a vector that gives the respective polynomial's coefficients in order of descending powers of s .

To design a filter, you can use functions such as `butter`, `cheby1`, and `cheby2` in the Signal Processing Toolbox. The default filter is a Chebyshev type II filter whose transfer function arises from the command below.

```
[num, den] = cheby2(3, 40, 100, 's')
```

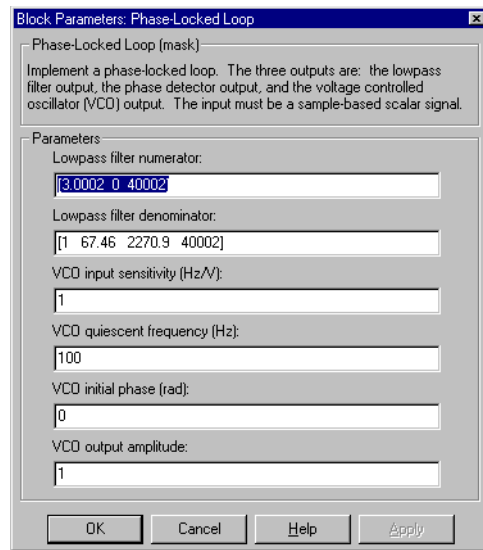
- A voltage-controlled oscillator (VCO). You specify characteristics of the VCO using the **VCO quiescent frequency**, **VCO initial phase**, and **VCO output amplitude** parameters.

The input signal represents the received signal. The input must be a sample-based scalar signal. The three output ports produce:

- The output of the filter
- The output of the phase detector
- The output of the VCO

Phase-Locked Loop

Dialog Box



The dialog box is titled "Block Parameters: Phase-Locked Loop". It contains a description of the block's function and several parameter input fields.

Phase-Locked Loop (mask):
Implement a phase-locked loop. The three outputs are: the lowpass filter output, the phase detector output, and the voltage controlled oscillator (VCO) output. The input must be a sample-based scalar signal.

Parameters:

- Lowpass filter numerator: [3.0002 0 40002]
- Lowpass filter denominator: [1 67.46 2270.9 40002]
- VCO input sensitivity (Hz/V): 1
- VCO quiescent frequency (Hz): 100
- VCO initial phase (rad): 0
- VCO output amplitude: 1

Buttons: OK, Cancel, Help, Apply

Lowpass filter numerator

The numerator of the lowpass filter's transfer function, represented as a vector that lists the coefficients in order of descending powers of s .

Lowpass filter denominator

The denominator of the lowpass filter's transfer function, represented as a vector that lists the coefficients in order of descending powers of s .

VCO input sensitivity (Hz/V)

This value scales the input to the VCO and, consequently, the shift from the **VCO quiescent frequency** value. The units of **VCO input sensitivity** are Hertz per volt.

VCO quiescent frequency (Hz)

The frequency of the VCO signal when the voltage applied to it is zero. This should match the carrier frequency of the input signal.

VCO initial phase (rad)

The initial phase of the VCO signal.

VCO output amplitude

The amplitude of the VCO signal.

See Also

Baseband PLL, Linearized Baseband PLL, Charge Pump PLL

References

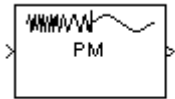
For more information about phase-locked loops, see the works listed in “Selected Bibliography for Synchronization” on page 2-92.

PM Demodulator Baseband

Purpose Demodulate PM-modulated data

Library Analog Baseband Modulation, in Modulation

Description

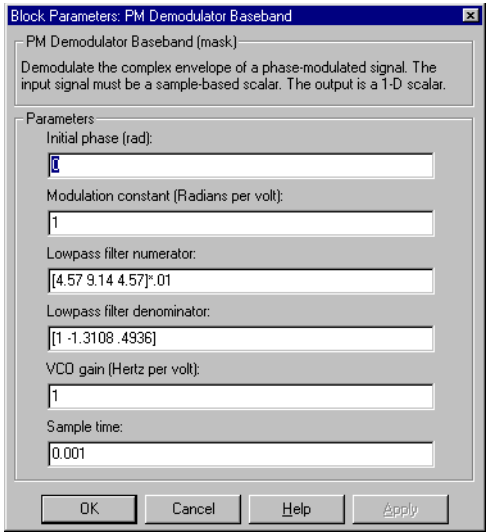


The PM Demodulator Baseband block demodulates a signal that was modulated using phase modulation. The input is a baseband representation of the modulated signal. The input is complex, while the output is real. The input must be a sample-based scalar signal.

This block uses a phase-locked loop containing a voltage-controlled oscillator (VCO). The **VCO Gain** parameter specifies the input sensitivity of the VCO.

In the course of demodulating, the block uses a filter whose transfer function is described by the **Lowpass filter numerator** and **Lowpass filter denominator** parameters.

Dialog Box



Initial phase (rad)

The initial phase in the corresponding PM Modulator Baseband block.

Modulation constant (Radians per volt)

The modulation constant in the corresponding PM Modulator Baseband block.

Lowpass filter numerator

The numerator of the lowpass filter transfer function. It is represented as a vector that lists the coefficients in order of descending powers of s .

Lowpass filter denominator

The denominator of the lowpass filter transfer function. It is represented as a vector that lists the coefficients in order of descending powers of s . For an FIR filter, set this parameter to 1.

VCO gain (Hertz per volt)

The input sensitivity of the voltage-controlled oscillator.

Sample time

The sample time of the output signal.

Pair Block

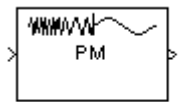
PM Modulator Baseband

PM Demodulator Passband

Purpose Demodulate PM-modulated data

Library Analog Passband Modulation, in Modulation

Description



The PM Demodulator Passband block demodulates a signal that was modulated using phase modulation. The input is a passband representation of the modulated signal. Both the input and output signals are real sample-based scalar signals.

This block uses a phase-locked loop containing a voltage-controlled oscillator (VCO). The **VCO Gain** parameter specifies the input sensitivity of the VCO.

In the course of demodulating, the block uses a filter whose transfer function is described by the **Lowpass filter numerator** and **Lowpass filter denominator** parameters.

By the Nyquist sampling theorem, the reciprocal of the **Sample time** parameter must exceed twice the **Carrier frequency** parameter.

Dialog Box

Block Parameters: PM Demodulator Passband

PM Demodulator Passband (mask)

Demodulate a phase modulated signal using a phase-locked loop. The input signal must be a sample-based scalar. The output is a 1-D scalar.

Parameters

Carrier frequency (Hz):
100

Initial phase (rad):
0

Modulation constant (Radians per volt):
1

Lowpass filter numerator:
[4.57 9.14 4.57]e-01

Lowpass filter denominator:
[1 -1.3108 .4936]

VCO Gain (Hertz per volt):
1

Sample time:
0.001

OK

Cancel

Help

Apply

4-350

Carrier frequency (Hz)

The carrier frequency in the corresponding PM Modulator Passband block.

Initial phase (rad)

The carrier signal's initial phase in the corresponding PM Modulator Passband block.

Modulation constant (Radians per volt)

The modulation constant in the corresponding PM Modulator Passband block.

Lowpass filter numerator

The numerator of the lowpass filter transfer function. It is represented as a vector that lists the coefficients in order of descending powers of s .

Lowpass filter denominator

The denominator of the lowpass filter transfer function. It is represented as a vector that lists the coefficients in order of descending powers of s . For an FIR filter, set this parameter to 1.

VCO Gain (Hertz per volt)

The input sensitivity of the voltage-controlled oscillator.

Sample time

The sample time of the output signal.

Pair Block

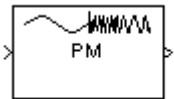
PM Modulator Passband

PM Modulator Baseband

Purpose Modulate using phase modulation

Library Analog Baseband Modulation, in Modulation

Description The PM Modulator Baseband block modulates using phase modulation. The output is a baseband representation of the modulated signal. The input signal is real, while the output signal is complex. The input must be a sample-based scalar signal.

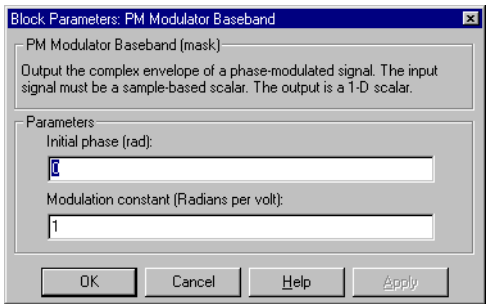


If the input is $u(t)$ as a function of time t , then the output is

$$\exp(j\theta + jK_c u(t))$$

where θ is the **Initial phase** parameter and K_c is the **Modulation constant** parameter.

Dialog Box



Initial phase (rad)
The phase of the modulated signal when the input is zero.

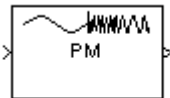
Modulation constant (Radians per volt)
Modulation constant K_c

Pair Block PM Demodulator Baseband

Purpose Modulate using phase modulation

Library Analog Passband Modulation, in Modulation

Description The PM Modulator Passband block modulates using phase modulation. The output is a passband representation of the modulated signal. The output signal's frequency varies with the input signal's amplitude. Both the input and output signals are real sample-based scalar signals.



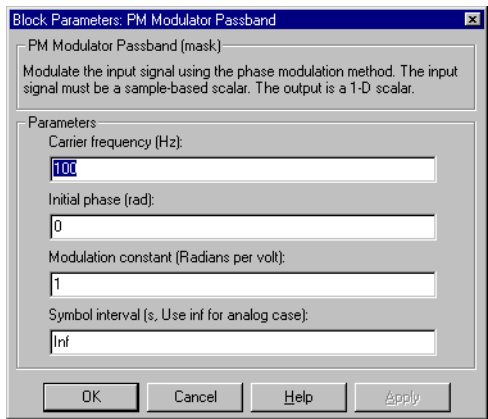
If the input is $u(t)$ as a function of time t , then the output is

$$\cos(2\pi f_c t + K_c u(t) + \theta)$$

where f_c is the **Carrier frequency** parameter, θ is the **Initial phase** parameter, and K_c is the **Modulation constant** parameter.

An appropriate **Carrier frequency** value is generally much higher than the highest frequency of the input signal. To avoid having to use a high carrier frequency and consequently a high sampling rate, you can use baseband simulation (PM Modulator Baseband block) instead of passband simulation.

Dialog Box



Carrier frequency (Hz)

The frequency of the carrier.

PM Modulator Passband

Initial phase (rad)

The initial phase of the carrier in radians.

Modulation constant (Radians per volt)

The modulation constant K_c .

Symbol interval

Inf by default. To use this block to model PSK, set this parameter to the length of time required to transmit a single information bit.

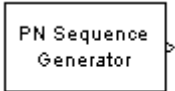
Pair Block PM Demodulator Passband

See Also PM Modulator Baseband

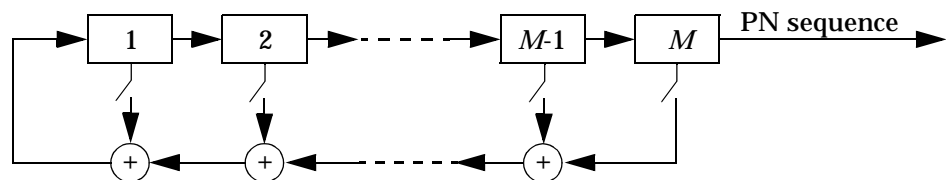
Purpose Generate pseudonoise sequence

Library Comm Sources

Description The PN Sequence Generator block generates a sequence of pseudorandom binary numbers. A pseudonoise sequence can be used in a pseudorandom scrambler and descrambler. It can also be used in a direct-sequence spread-spectrum system.



Below is a schematic of the pseudorandom sequence generator. All adders perform addition modulo 2.



All M registers in the generator update their values at each time step. The state of each switch is defined by the generator polynomial, which is a polynomial in z^{-1} having binary coefficients. The constant term of the generator polynomial must be 1. You can specify the **Generator polynomial** parameter using either of these formats:

- A vector of coefficients of the polynomial, starting with the constant term and proceeding in order of increasing powers of z^{-1} . The first and last elements must be 1.
- A vector containing the exponents of z (not z^{-1}) for the nonzero terms of the polynomial. The first element must be zero.

For example, $p = [1 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1]$ and $p = [0 \ -6 \ -8]$ represent the same polynomial $p(z) = 1 + z^{-6} + z^{-8}$.

It is very important that the **Initial states** parameter have a suitable value. The initial state of at least one of the registers must be nonzero in order to generate a nonzero sequence. The length of the **Initial states** vector must equal the order of the generator polynomial, and its elements must be binary numbers. If the **Generator polynomial** parameter is a vector that lists the

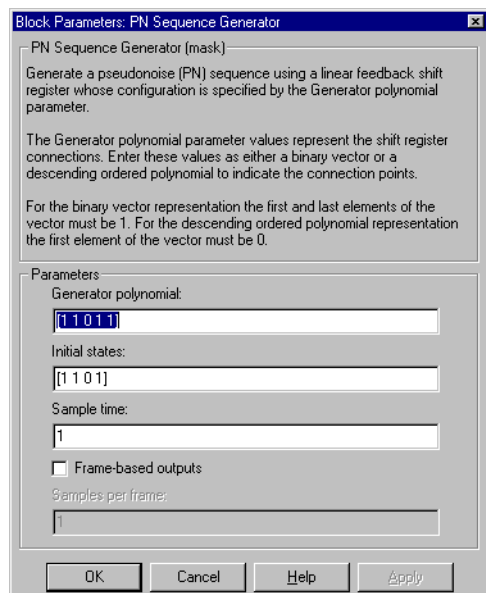
PN Sequence Generator

coefficients in order, then the order of the generator polynomial is one less than the vector length.

Attributes of Output Signal

If the **Frame-based outputs** box is checked, then the output signal is a frame-based column vector whose length is the **Samples per frame** parameter. Otherwise, the output signal is a one-dimensional scalar.

Dialog Box



Generator polynomial

A polynomial that determines the shift register's feedback connections.

Initial states

A vector of initial states of the shift registers.

Sample time

The period of each element of the output signal.

Frame-based outputs

Determines whether the output is frame-based or sample-based.

Samples per frame

The number of samples in a frame-based output signal. This field is active only if **Frame-based outputs** is checked.

See Also

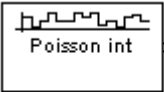
Scrambler

Poisson Int Generator

Purpose Generate Poisson-distributed random integers

Library Comm Sources

Description



The Poisson Int Generator block generates random integers using a Poisson distribution. The probability of generating a nonnegative integer k is $\lambda^k \exp(-\lambda) / (k!)$, where λ is a positive number known as the Poisson parameter.

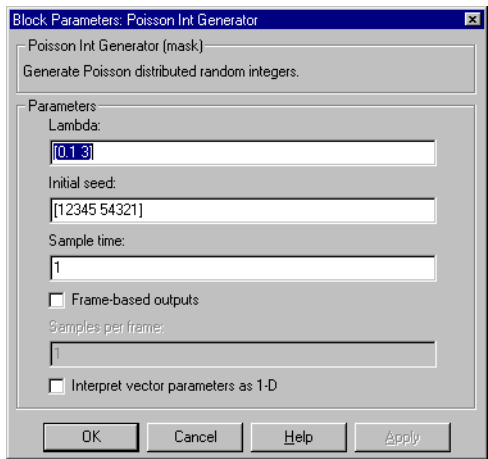
You can use the Poisson Int Generator to generate noise in a binary transmission channel. In this case, the Poisson parameter **Lambda** should be less than 1, usually much less.

Attributes of Output Signal

The output signal can be a frame-based matrix, a sample-based row or column vector, or a sample-based one-dimensional array. These attributes are controlled by the **Frame-based outputs**, **Samples per frame**, and **Interpret vector parameters as 1-D** parameters. See “Signal Attribute Parameters for Random Sources” on page 2-7 for more details.

The number of elements in the **Initial seed** parameter becomes the number of columns in a frame-based output or the number of elements in a sample-based vector output. Also, the shape (row or column) of the **Initial seed** parameter becomes the shape of a sample-based two-dimensional output signal.

Dialog Box



Lambda

The Poisson parameter λ . If it is a scalar, then every element in the output vector shares the same Poisson parameter.

Initial seed

The initial seed value for the random number generator.

Sample time

The period of each sample-based vector or each row of a frame-based matrix.

Frame-based outputs

Determines whether the output is frame-based or sample-based. This box is active only if **Interpret vector parameters as 1-D** is unchecked.

Samples per frame

The number of samples in each column of a frame-based output signal. This field is active only if **Frame-based outputs** is checked.

Interpret vector parameters as 1-D

If this box is checked, then the output is a one-dimensional signal. Otherwise, the output is a two-dimensional signal. This box is active only if **Frame-based outputs** is unchecked.

See Also

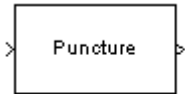
Random-Integer Generator; `poi ssrnd` (Statistics Toolbox)

Puncture

Purpose Output the elements which correspond to 1s in the binary Puncture vector

Library Sequence Operations, in Basic Comm Functions

Description



The Puncture block creates an output vector by removing selected elements of the input vector and preserving others. The input can be a real or complex vector of length K. The block determines which elements to remove or preserve by using the binary **Puncture vector** parameter:

- If **Puncture vector**(k) = 0, then the kth element of the input vector does not become part of the output vector.
- If **Puncture vector**(k) = 1, then the kth element of the input vector is preserved in the output vector.

Here, k is between 1 and K. The preserved elements appear in the output vector in the same order in which they appear in the input vector.

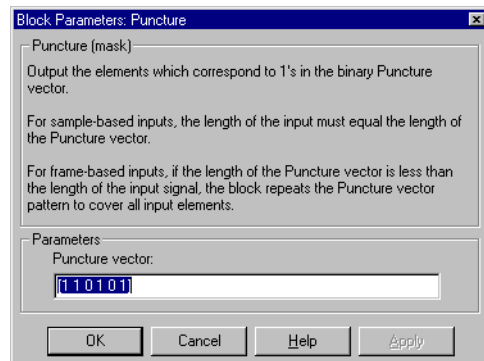
Frame-Based Processing

If the input is frame-based, then both it and the **Puncture vector** parameter must be column vectors. The length of the **Puncture vector** parameter must divide K. The block repeats the puncturing pattern, if necessary, to cover all input elements. That is, in the bulleted items above you can replace **Puncture vector**(k) by **Puncture vector**(n), where

$$n = \text{mod}(k, \text{length}(\text{Puncture vector}))$$

and mod is the modulus function (mod in MATLAB).

Dialog Box



Puncture vector

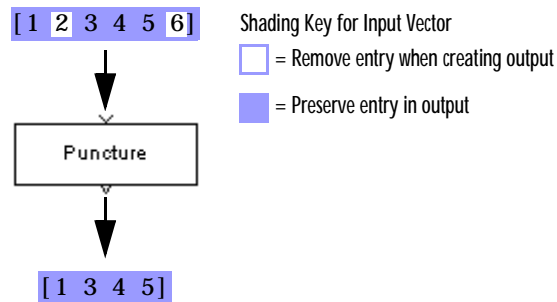
A binary vector whose pattern of 0s (1s) indicates which elements of the input the block should remove (preserve).

Examples

If the **Puncture vector** parameter is the six-element vector [1; 0; 1; 1; 1; 0], then the block:

- Removes the second and sixth elements from the group of six input elements.
- Sends the first, third, fourth, and fifth elements to the output vector.

The diagram below depicts the block's operation on an input vector of [1: 6], using this **Puncture vector** parameter.



See Also

Insert Zero

QPSK Demodulator Baseband

Purpose Demodulate QPSK-modulated data

Library PM, in Digital Baseband sublibrary of Modulation

Description



The QPSK Demodulator Baseband block demodulates a signal that was modulated using the quaternary phase shift keying method. The input is a baseband representation of the modulated signal.

The input must be a discrete-time complex signal. The input can be either a scalar or a frame-based column vector.

If the **Output type** parameter is set to **Integer**, then the block maps the point $\exp(j\theta + j\pi m/2)$

to m , where θ is the **Phase offset** parameter and m is 0, 1, 2, or 3.

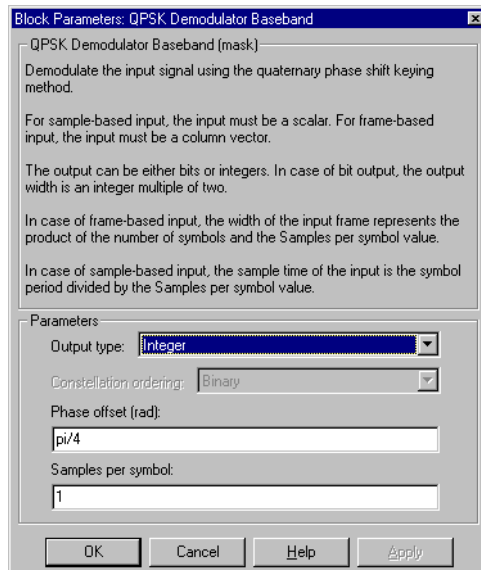
If the **Output type** parameter is set to **Bit**, then the output contains pairs of binary values. The reference page for the QPSK Modulator Baseband block shows the signal constellations for the cases when the **Constellation ordering** parameter is either **Binary** or **Gray**.

Processing an Upsampled Modulated Signal

The input signal can be an upsampled version of the modulated signal. The **Samples per symbol** parameter is the upsampling factor. It must be a positive integer.

For more information, see “Upsampled Signals and Rate Changes” on page 2-72.

Dialog Box



Output type

Determines whether the output consists of integers or pairs of bits.

Constellation ordering

Determines how the block maps each integer to a pair of output bits. This field is active only when **Output type** is set to **Bit**.

Phase offset (rad)

The phase of the zeroth point of the signal constellation.

Samples per symbol

The number of input samples that represent each modulated symbol.

Pair Block

QPSK Demodulator Baseband

See Also

M-PSK Demodulator Baseband, BPSK Modulator Baseband, DQPSK Demodulator Baseband

QPSK Modulator Baseband

Purpose Modulate using the quaternary phase shift keying method

Library PM, in Digital Baseband sublibrary of Modulation

Description The QPSK Modulator Baseband block modulates using the quaternary phase shift keying method. The output is a baseband representation of the modulated signal.



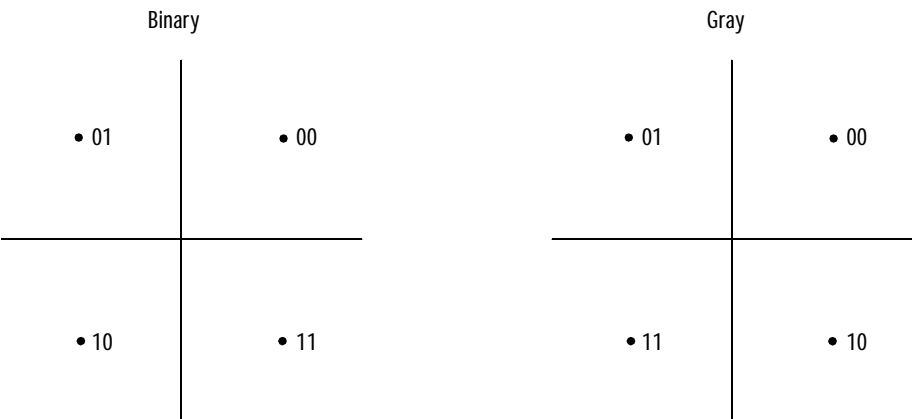
Inputs and Constellation Types

If the **Input type** parameter is set to **Integer**, then valid input values are 0, 1, 2, and 3. If the input is m , then the output symbol is

$$\exp(j\theta + j\pi m/2)$$

where θ is the **Phase offset** parameter. In this case, the input can be either a scalar or a frame-based column vector.

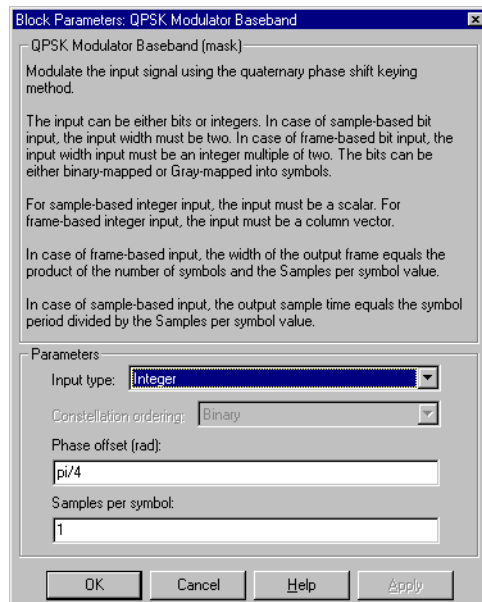
If the **Input type** parameter is set to **Bit**, then the input contains pairs of binary values. The input can be either a vector of length two or a frame-based column vector whose length is an even integer. If the **Phase offset** parameter is set to $\pi/4$, then the block uses one of the signal constellations in the figure below, depending on whether the **Constellation ordering** parameter is set to **Binary** or **Gray**.



Upsampling the Modulated Signal

This block can output an upsampled version of the modulated signal. The **Samples per symbol** parameter is the upsampling factor. It must be a positive integer. For more information, see “Upsampled Signals and Rate Changes” on page 2-72.

Dialog Box



Input type

Indicates whether the input consists of integers or pairs of bits.

Constellation ordering

Determines how the block maps each pair of input bits to a corresponding integer. This field is active only when **Input type** is set to **Bit**.

Phase offset (rad)

The phase of the zeroth point of the signal constellation.

Samples per symbol

The number of output samples that the block produces for each integer or pair of bits in the input.

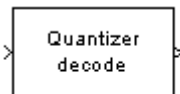
QPSK Modulator Baseband

Pair Block	QPSK Demodulator Baseband
See Also	M-PSK Modulator Baseband, BPSK Modulator Baseband, DQPSK Modulator Baseband

Purpose Decode quantization index according to codebook

Library Source Coding

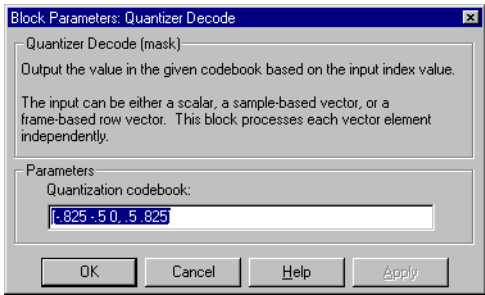
Description The Quantizer Decode block recovers a message from a quantized signal, converting the quantization index into the corresponding codebook value. The **Quantization codebook** parameter, a vector of length N, prescribes the possible output values. If the input is an integer k between 0 and N-1, then the output is the $(k+1)$ st element of **Quantization codebook**.



The input can be either a scalar, a sample-based vector, or a frame-based row vector. This block processes each vector element independently. Each output signal is a vector of the same length as the input signal.

Note The Sampled Quantizer Encode and Enabled Quantizer Encode blocks also use a **Quantization codebook** parameter. The first output of those blocks corresponds to the input of Quantizer Decode; the second output of those blocks corresponds to the output of Quantizer Decode.

Dialog Box



Quantization codebook

A real vector that prescribes the output value corresponding to each input integer.

Pair Block Sampled Quantizer Encode or Enabled Quantizer Encode

Random Deinterleaver

Purpose Restore ordering of the input symbols using a random permutation

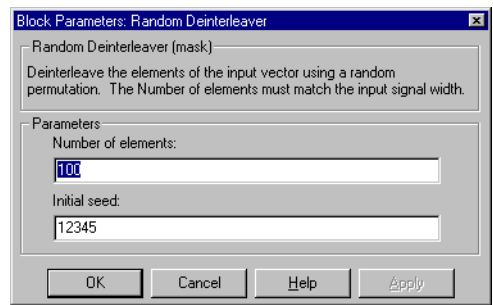
Library Block sublibrary of Interleaving

Description The Random Deinterleaver block rearranges the elements of its input vector using a random permutation. The **Initial seed** parameter initializes the random number generator that the block uses to determine the permutation. If this block and the Random Interleaver block have the same value for **Initial seed**, then the two blocks are inverses of each other.



The **Number of elements** parameter indicates how many numbers are in the input vector. If the input is frame-based, then it must be a column vector.

Dialog Box



Number of elements
The number of elements in the input vector.

Initial seed
The initial seed value for the random number generator.

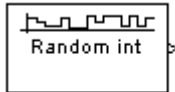
Pair Block Random Interleaver

See Also General Block Deinterleaver

Purpose Generate integers randomly distributed in the range $[0, M-1]$

Library Comm Sources

Description



The Random-Integer Generator block generates uniformly distributed random integers in the range $[0, M-1]$, where M is the **M-ary number** defined in the dialog box.

The **M-ary number** can be either a scalar or a vector. If it is a scalar, then all output random variables are independent and identically distributed (i.i.d.). If the **M-ary number** is a vector, then its length must equal the length of the **Initial seed**; in this case each output has its own output range.

If the **Initial seed** parameter is a constant, then the resulting noise is repeatable.

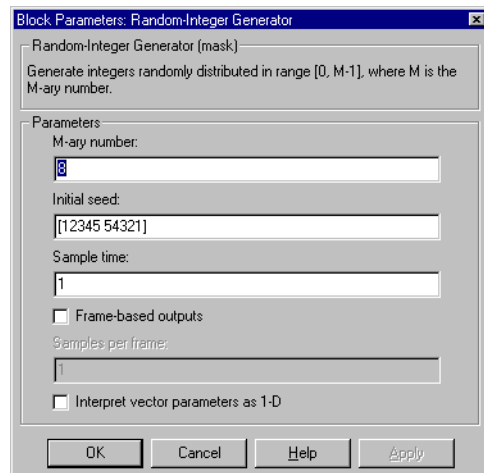
Attributes of Output Signal

The output signal can be a frame-based matrix, a sample-based row or column vector, or a sample-based one-dimensional array. These attributes are controlled by the **Frame-based outputs**, **Samples per frame**, and **Interpret vector parameters as 1-D** parameters. See “Signal Attribute Parameters for Random Sources” on page 2-7 for more details.

The number of elements in the **Initial seed** parameter becomes the number of columns in a frame-based output or the number of elements in a sample-based vector output. Also, the shape (row or column) of the **Initial seed** parameter becomes the shape of a sample-based two-dimensional output signal.

Random-Integer Generator

Dialog Box



M-ary number

The positive integer, or vector of positive integers, that indicates the range of output values.

Initial seed

The initial seed value for the random number generator. The vector length of the seed determines the length of the output vector.

Sample time

The period of each sample-based vector or each row of a frame-based matrix.

Frame-based outputs

Determines whether the output is frame-based or sample-based. This box is active only if **Interpret vector parameters as 1-D** is unchecked.

Samples per frame

The number of samples in each column of a frame-based output signal. This field is active only if **Frame-based outputs** is checked.

Interpret vector parameters as 1-D

If this box is checked, then the output is a one-dimensional signal.

Otherwise, the output is a two-dimensional signal. This box is active only if **Frame-based outputs** is unchecked.

See Also

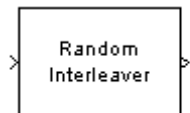
`randi nt` (Communications Toolbox)

Random Interleaver

Purpose Reorder the input symbols using a random permutation

Library Block sublibrary of Interleaving

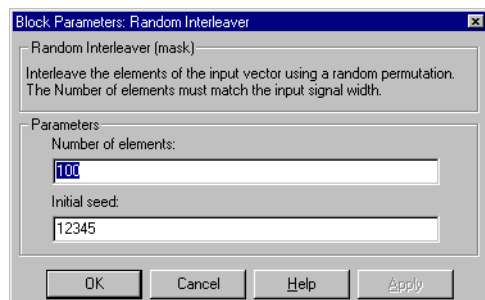
Description



The Random Interleaver block rearranges the elements of its input vector using a random permutation. The **Number of elements** parameter indicates how many numbers are in the input vector. If the input is frame-based, then it must be a column vector.

The **Initial seed** parameter initializes the random number generator that the block uses to determine the permutation. The block is predictable for a given seed, but different seeds produce different permutations.

Dialog Box



Number of elements

The number of elements in the input vector.

Initial seed

The initial seed value for the random number generator.

Pair Block Random Deinterleaver

See Also General Block Interleaver

Purpose Generate Rayleigh distributed noise

Library Comm Sources

Description The Rayleigh Noise Generator block generates Rayleigh distributed noise. The Rayleigh probability density function is given by



$$f(x) = \begin{cases} \frac{x}{\sigma^2} e^{-\frac{x^2}{2\sigma^2}} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

where σ^2 is known as the *fading envelope* of the Rayleigh distribution.

The block requires you to specify the **Initial seed** for the random number generator. If it is a constant, then the resulting noise is repeatable. The **sigma** parameter can be either a vector of the same length as the **Initial seed**, or a scalar. When **sigma** is a scalar, every element of the output signal shares that same value.

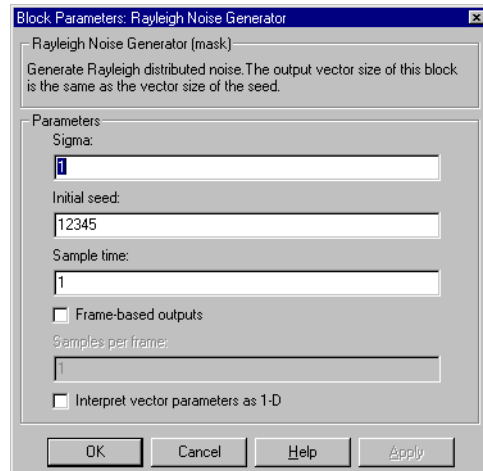
Attributes of Output Signal

The output signal can be a frame-based matrix, a sample-based row or column vector, or a sample-based one-dimensional array. These attributes are controlled by the **Frame-based outputs**, **Samples per frame**, and **Interpret vector parameters as 1-D** parameters. See “Signal Attribute Parameters for Random Sources” on page 2-7 for more details.

The number of elements in the **Initial seed** parameter becomes the number of columns in a frame-based output or the number of elements in a sample-based vector output. Also, the shape (row or column) of the **Initial seed** parameter becomes the shape of a sample-based two-dimensional output signal.

Rayleigh Noise Generator

Dialog Box



Sigma

Specify σ as defined in the Rayleigh probability density function.

Initial seed

The initial seed value for the random number generator.

Sample time

The period of each sample-based vector or each row of a frame-based matrix.

Frame-based outputs

Determines whether the output is frame-based or sample-based. This box is active only if **Interpret vector parameters as 1-D** is unchecked.

Samples per frame

The number of samples in each column of a frame-based output signal. This field is active only if **Frame-based outputs** is checked.

Interpret vector parameters as 1-D

If this box is checked, then the output is a one-dimensional signal. Otherwise, the output is a two-dimensional signal. This box is active only if **Frame-based outputs** is unchecked.

See Also Multipath Rayleigh Fading Channel; rayl rnd (Statistics Toolbox)

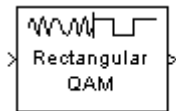
References [1] Proakis, John G. *Digital Communications*, Third edition. New York: McGraw Hill, 1995.

Rectangular QAM Demodulator Baseband

Purpose Demodulate QAM-modulated data

Library AM, in Digital Baseband sublibrary of Modulation

Description



The Rectangular QAM Demodulator Baseband block demodulates a signal that was modulated using quadrature amplitude modulation with a constellation on a rectangular lattice.

The signal constellation has M points, where M is the **M-ary number** parameter. M must have the form 2^K for some positive integer K . The block scales the signal constellation based on how you set the **Normalization method** parameter. For details, see the reference page for the Rectangular QAM Modulator Baseband block.

The input can be either a scalar or a frame-based column vector.

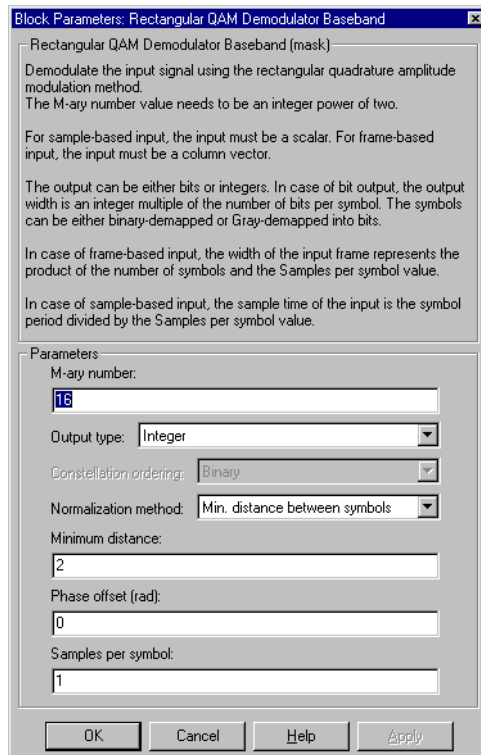
Output Signal Values

The **Output type** parameter determines whether the block produces integers or binary representations of integers. If **Output type** is set to **Integer**, then the block produces integers. If **Output type** is set to **Bit**, then the block produces a group of K bits, called a binary word, for each symbol. The **Constellation ordering** parameter indicates how the block assigns binary words to points of the signal constellation. More details are on the reference page for the Rectangular QAM Modulator Baseband block.

Processing an Upsampled Modulated Signal

The input signal can be an upsampled version of the modulated signal. The **Samples per symbol** parameter is the upsampling factor. It must be a positive integer. For more information, see “Upsampled Signals and Rate Changes” on page 2-72.

Dialog Box



Block Parameters: Rectangular QAM Demodulator Baseband

Rectangular QAM Demodulator Baseband (mask)

Demodulate the input signal using the rectangular quadrature amplitude modulation method.
The M-ary number value needs to be an integer power of two.

For sample-based input, the input must be a scalar. For frame-based input, the input must be a column vector.

The output can be either bits or integers. In case of bit output, the output width is an integer multiple of the number of bits per symbol. The symbols can be either binary-demapped or Gray-demapped into bits.

In case of frame-based input, the width of the input frame represents the product of the number of symbols and the Samples per symbol value.

In case of sample-based input, the sample time of the input is the symbol period divided by the Samples per symbol value.

Parameters

M-ary number:

Output type:

Constellation ordering:

Normalization method:

Minimum distance:

Phase offset (rad):

Samples per symbol:

OK Cancel Help Apply

M-ary number

The number of points in the signal constellation. It must have the form 2^K for some positive integer K.

Output type

Indicates whether the output consists of integers or groups of bits.

Constellation ordering

Determines how the block maps each integer to a group of output bits. This field is active only when **Output type** is set to **Bit**.

Normalization method

Determines how the block scales the signal constellation. Choices are **Min. distance between symbols**, **Average Power**, and **Peak Power**.

Rectangular QAM Demodulator Baseband

Minimum distance

The distance between two nearest constellation points. This field appears only when **Normalization method** is set to **Min. distance between symbols**.

Average power (watts)

The average power of the symbols in the constellation. This field appears only when **Normalization method** is set to **Average Power**.

Peak power (watts)

The maximum power among the symbols in the constellation. This field appears only when **Normalization method** is set to **Peak Power**.

Phase offset (rad)

The rotation of the signal constellation, in radians.

Samples per symbol

The number of input samples that represent each modulated symbol.

Pair Block Rectangular QAM Modulator Baseband

See Also General QAM Demodulator Baseband

References [1] Smith, Joel G. "Odd-Bit Quadrature Amplitude-Shift Keying." *IEEE Transactions on Communications*, vol. COM-23, March 1975. 385-389.

Rectangular QAM Demodulator Passband

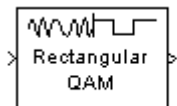
Purpose

Demodulate QAM-modulated data

Library

AM, in Digital Passband sublibrary of Modulation

Description



The Rectangular QAM Demodulator Passband block demodulates a signal that was modulated using quadrature amplitude modulation with a constellation on a rectangular lattice. The signal constellation has M points, where M is the **M-ary number** parameter. M must have the form 2^K for some positive integer K .

This block converts the input to an equivalent baseband representation and then uses the baseband equivalent block, M-PAM Demodulator Baseband, for internal computations. The following parameters in this block are the same as those of the baseband equivalent block:

- **M-ary number**
- **Output type**
- **Constellation ordering**
- **Normalization method**
- **Minimum distance**
- **Average power**
- **Peak power**

The input must be a sample-based scalar signal.

Parameters Specific to Passband Simulation

Passband simulation uses a carrier signal. The **Carrier frequency** and **Carrier initial phase** parameters specify the frequency and initial phase, respectively, of the carrier signal. The **Input sample time** parameter specifies the sample time of the input signal, while the **Symbol period** parameter equals the sample time of the output signal.

This block uses a baseband representation of the modulated signal as an intermediate signal during internal computations. The **Baseband samples per symbol** parameter indicates how many baseband samples correspond to each integer or binary word in the output.

The timing-related parameters must satisfy these relationships:

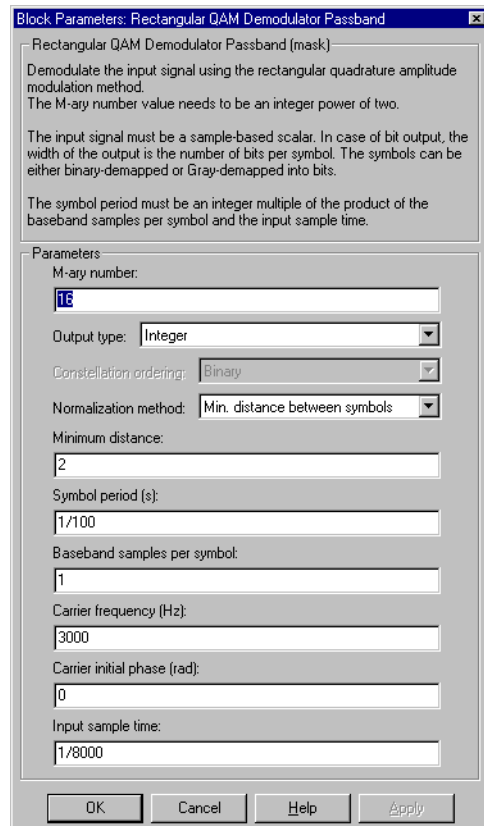
- **Input sample time** $> (\text{Carrier frequency})^{-1}$

Rectangular QAM Demodulator Passband

- **Symbol period** $< [2 * \text{Carrier frequency} + 2 / (\text{Input sample time})]^{-1}$
- **Input sample time** = $K * \text{Symbol period} * \text{Baseband samples per symbol}$ for some integer K

Also, this block incurs an extra output period of delay compared to its baseband equivalent block.

Dialog Box



The dialog box is titled "Block Parameters: Rectangular QAM Demodulator Passband". It contains a description of the block's function and a list of parameters to be configured.

Rectangular QAM Demodulator Passband (mask)
Demodulate the input signal using the rectangular quadrature amplitude modulation method.
The M-ary number value needs to be an integer power of two.

The input signal must be a sample-based scalar. In case of bit output, the width of the output is the number of bits per symbol. The symbols can be either binary-demapped or Gray-demapped into bits.

The symbol period must be an integer multiple of the product of the baseband samples per symbol and the input sample time.

Parameters

M-ary number:

Output type:

Constellation ordering:

Normalization method:

Minimum distance:

Symbol period (s):

Baseband samples per symbol:

Carrier frequency (Hz):

Carrier initial phase (rad):

Input sample time:

Buttons: OK, Cancel, Help, Apply

M-ary number

The number of points in the signal constellation. It must have the form 2^K for some positive integer K.

Output type

Indicates whether the output consists of integers or groups of bits.

Constellation ordering

Determines how the block maps each integer to a group of output bits. This field is active only when **Output type** is set to **Bit**.

Normalization method

Determines how the block scales the signal constellation. Choices are **Min. distance between symbols**, **Average Power**, and **Peak Power**.

Minimum distance

The distance between two nearest constellation points. This field appears only when **Normalization method** is set to **Min. distance between symbols**.

Average power (watts)

The average power of the symbols in the constellation. This field appears only when **Normalization method** is set to **Average Power**.

Peak power (watts)

The maximum power among the symbols in the constellation. This field appears only when **Normalization method** is set to **Peak Power**.

Symbol period (s)

The symbol period, which equals the sample time of the output.

Baseband samples per symbol

The number of baseband samples that represent each modulated symbol, after the block converts the passband input to a baseband intermediary signal.

Carrier frequency (Hz)

The frequency of the carrier.

Carrier initial phase (rad)

The initial phase of the carrier in radians.

Input sample time

The sample time of the input signal.

Rectangular QAM Demodulator Passband

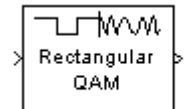
Pair Block	Rectangular QAM Modulator Passband
See Also	General QAM Demodulator Passband, Rectangular QAM Demodulator Baseband
References	[1] Smith, Joel G. "Odd-Bit Quadrature Amplitude-Shift Keying." <i>IEEE Transactions on Communications</i> , vol. COM-23, March 1975. 385-389.

Rectangular QAM Modulator Baseband

Purpose Modulate using M-ary quadrature amplitude modulation

Library AM, in Digital Baseband sublibrary of Modulation

Description The Rectangular QAM Modulator Baseband block modulates using M-ary quadrature amplitude modulation with a constellation on a rectangular lattice. The output is a baseband representation of the modulated signal.



Constellation Size and Scaling

The signal constellation has M points, where M is the **M-ary number** parameter. M must have the form 2^K for some positive integer K. The block scales the signal constellation based on how you set the **Normalization method** parameter. The table below lists the possible scaling conditions.

Value of Normalization method parameter	Scaling Condition
Min. distance between symbols	The nearest pair of points in the constellation is separated by the value of the Minimum distance parameter.
Average Power	The average power of the symbols in the constellation is the Average power parameter.
Peak Power	The maximum power of the symbols in the constellation is the Peak power parameter.

Input Signal Values

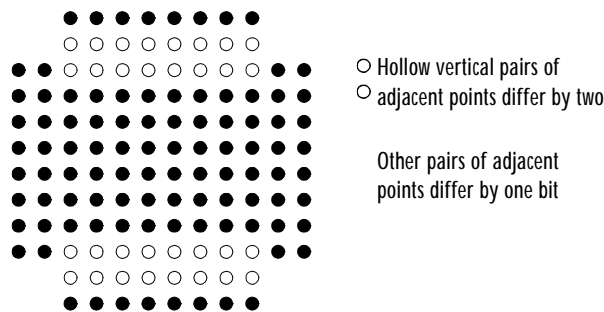
The input and output for this block are discrete-time signals. The **Input type** parameter determines whether the block accepts integers between 0 and M-1, or binary representations of integers:

- If **Input type** is set to **Integer**, then the block accepts integers. The input can be either a scalar or a frame-based column vector.
- If **Input type** is set to **Bit**, then the block accepts groups of K bits, called binary words. The input can be either a vector of length K or a frame-based

Rectangular QAM Modulator Baseband

column vector whose length is an integer multiple of K. The **Constellation ordering** parameter indicates how the block assigns binary words to points of the signal constellation. Such assignments apply independently to the in-phase and quadrature components of the input:

- If **Constellation ordering** is set to **Binary**, then the block uses a natural binary-coded constellation.
- If **Constellation ordering** is set to **Gray** and K is even, then the block uses a Gray-coded constellation.
- If **Constellation ordering** is set to **Gray** and K is odd, then the block codes the constellation so that pairs of nearest points differ in one or two bits. The constellation is cross-shaped, and the schematic below indicates which pairs of points differ in two bits. The schematic uses $M = 128$, but suggests the general case.

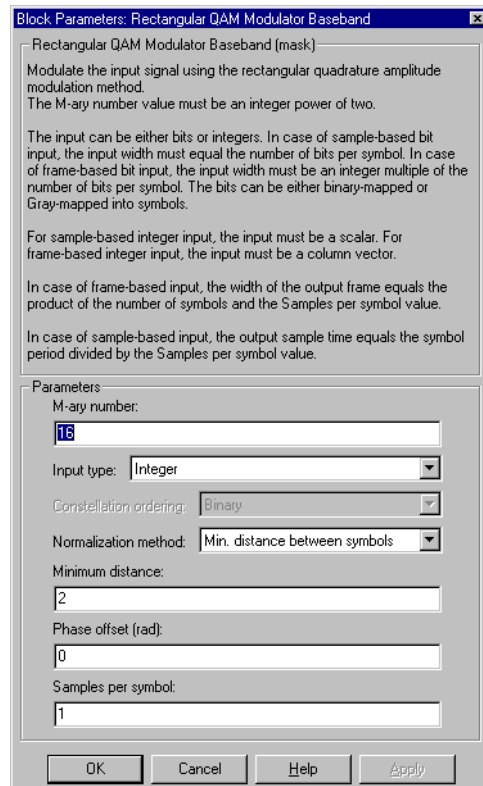


For details about the Gray coding, see the reference page for the M-PSK Modulator Baseband block and the paper listed in “References” below. Note that since the in-phase and quadrature components are assigned independently, the Gray and binary orderings coincide when $M = 4$.

Upsampling the Modulated Signal

This block can output an upsampled version of the modulated signal. The **Samples per symbol** parameter is the upsampling factor. It must be a positive integer. For more information, see “Upsampled Signals and Rate Changes” on page 2-72.

Dialog Box



Block Parameters: Rectangular QAM Modulator Baseband

Rectangular QAM Modulator Baseband (mask)

Modulate the input signal using the rectangular quadrature amplitude modulation method.
The M-ary number value must be an integer power of two.

The input can be either bits or integers. In case of sample-based bit input, the input width must equal the number of bits per symbol. In case of frame-based bit input, the input width must be an integer multiple of the number of bits per symbol. The bits can be either binary-mapped or Gray-mapped into symbols.

For sample-based integer input, the input must be a scalar. For frame-based integer input, the input must be a column vector.

In case of frame-based input, the width of the output frame equals the product of the number of symbols and the Samples per symbol value.

In case of sample-based input, the output sample time equals the symbol period divided by the Samples per symbol value.

Parameters

M-ary number:

Input type:

Constellation ordering:

Normalization method:

Minimum distance:

Phase offset (rad):

Samples per symbol:

OK Cancel Help Apply

M-ary number

The number of points in the signal constellation. It must have the form 2^K for some positive integer K .

Input type

Indicates whether the input consists of integers or groups of bits.

Constellation ordering

Determines how the block maps each group of input bits to a corresponding integer. This field is active only when **Input type** is set to **Bit**.

Rectangular QAM Modulator Baseband

Normalization method

Determines how the block scales the signal constellation. Choices are **Min. distance between symbols**, **Average Power**, and **Peak Power**.

Minimum distance

The distance between two nearest constellation points. This field appears only when **Normalization method** is set to **Min. distance between symbols**.

Average power (watts)

The average power of the symbols in the constellation. This field appears only when **Normalization method** is set to **Average Power**.

Peak power (watts)

The maximum power of the symbols in the constellation. This field appears only when **Normalization method** is set to **Peak Power**.

Phase offset (rad)

The rotation of the signal constellation, in radians.

Samples per symbol

The number of output samples that the block produces for each integer or binary word in the input.

Pair Block Rectangular QAM Demodulator Baseband

See Also General QAM Modulator Baseband

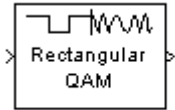
References [1] Smith, Joel G. "Odd-Bit Quadrature Amplitude-Shift Keying." *IEEE Transactions on Communications*, vol. COM-23, March 1975. 385-389.

Rectangular QAM Modulator Passband

Purpose Modulate using M-ary quadrature amplitude modulation

Library AM, in Digital Passband sublibrary of Modulation

Description



The Rectangular QAM Modulator Passband block modulates using M-ary quadrature amplitude modulation with a constellation on a rectangular lattice. The output is a passband representation of the modulated signal. The signal constellation has M points, where M is the **M-ary number** parameter. M must have the form 2^K for some positive integer K.

This block uses the baseband equivalent block, Rectangular QAM Modulator Baseband, for internal computations and converts the resulting baseband signal to a passband representation. The following parameters in this block are the same as those of the baseband equivalent block:

- **M-ary number**
- **Input type**
- **Constellation ordering**
- **Normalization method**
- **Minimum distance**
- **Average power**
- **Peak power**

The input must be sample-based. If the **Input type** parameter is **Bit**, then the input must be a vector of length $\log_2(M)$. If the **Input type** parameter is **Integer**, then the input must be a scalar.

Parameters Specific to Passband Simulation

Passband simulation uses a carrier signal. The **Carrier frequency** and **Carrier initial phase** parameters specify the frequency and initial phase, respectively, of the carrier signal. The **Symbol period** parameter must equal the sample time of the input signal, while the **Output sample time** parameter determines the sample time of the output signal.

This block uses a baseband representation of the modulated signal as an intermediate result during internal computations. The **Baseband samples per symbol** parameter indicates how many baseband samples correspond to each integer or binary word in the input, before the block converts them to a passband output.

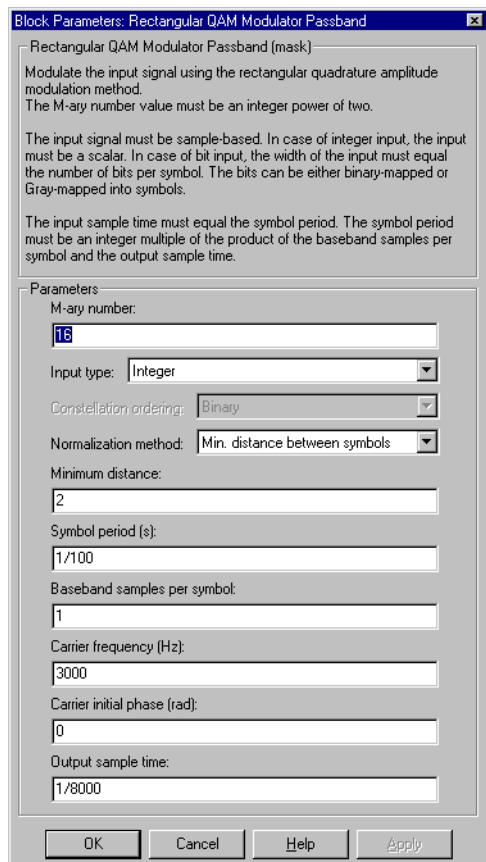
Rectangular QAM Modulator Passband

The timing-related parameters must satisfy these relationships:

- **Symbol period** > (**Carrier frequency**)⁻¹
- **Output sample time** < [2***Carrier frequency** + 2/(**Symbol period**)]⁻¹
- **Symbol period** = K***Output sample time*****Baseband samples per symbol**
for some integer K

Furthermore, **Carrier frequency** is typically much larger than the highest frequency of the unmodulated signal.

Dialog Box



Block Parameters: Rectangular QAM Modulator Passband

Rectangular QAM Modulator Passband (mask)

Modulate the input signal using the rectangular quadrature amplitude modulation method.
The M-ary number value must be an integer power of two.

The input signal must be sample-based. In case of integer input, the input must be a scalar. In case of bit input, the width of the input must equal the number of bits per symbol. The bits can be either binary-mapped or Gray-mapped into symbols.

The input sample time must equal the symbol period. The symbol period must be an integer multiple of the product of the baseband samples per symbol and the output sample time.

Parameters:

M-ary number:

Input type:

Constellation ordering:

Normalization method:

Minimum distance:

Symbol period (s):

Baseband samples per symbol:

Carrier frequency (Hz):

Carrier initial phase (rad):

Output sample time:

OK Cancel Help Apply

M-ary number

The number of points in the signal constellation. It must have the form 2^K for some positive integer K.

Input type

Indicates whether the input consists of integers or groups of bits.

Constellation ordering

Determines how the block maps each group of input bits to a corresponding integer. This field is active only when **Input type** is set to **Bit**.

Normalization method

Determines how the block scales the signal constellation. Choices are **Min. distance between symbols**, **Average Power**, and **Peak Power**.

Minimum distance

The distance between two nearest constellation points. This field appears only when **Normalization method** is set to **Min. distance between symbols**.

Average power (watts)

The average power of the symbols in the constellation. This field appears only when **Normalization method** is set to **Average Power**.

Peak power (watts)

The maximum power of the symbols in the constellation. This field appears only when **Normalization method** is set to **Peak Power**.

Symbol period (s)

The symbol period, which must equal the sample time of the input.

Baseband samples per symbol

The number of baseband samples that correspond to each integer or binary word in the input, before the block converts them to a passband output.

Carrier frequency (Hz)

The frequency of the carrier.

Carrier initial phase (rad)

The initial phase of the carrier in radians.

Rectangular QAM Modulator Passband

Output sample time
The sample time of the output signal.

Pair Block Rectangular QAM Demodulator Passband

See Also General QAM Modulator Passband, Rectangular QAM Modulator Baseband

References [1] Smith, Joel G. “Odd-Bit Quadrature Amplitude-Shift Keying.” *IEEE Transactions on Communications*, vol. COM-23, March 1975. 385-389.

Purpose Simulate a Rician fading propagation channel

Library Channels

Description



The Rician Fading Channel block implements a baseband simulation of a Rician fading propagation channel. This block is useful for modeling mobile wireless communication systems when the transmitted signal can travel to the receiver along a dominant line-of-sight or direct path. If the signal can travel along a line-of-sight path and also along other fading paths, then you can use this block in parallel with the Multipath Rayleigh Fading Channel block. For details about fading channels, see the works listed in “References” on page 4-393.

The input can be either a scalar or a frame-based column vector. The input is a complex signal.

Fading causes the signal to spread and become diffuse. The **K-factor** parameter, which is part of the statistical description of the Rician distribution, represents the ratio between direct-path (unspread) power and diffuse power. The ratio is expressed linearly, not in decibels. While the **Gain** parameter controls the overall gain through the channel, the **K-factor** parameter controls the gain’s partition into direct and diffuse components.

Relative motion between the transmitter and receiver causes Doppler shifts in the signal frequency. The **Doppler frequency** parameter is the *maximum* Doppler shift that the signal undergoes. The Jakes PSD (power spectral density) determines the spectrum of the Rician process.

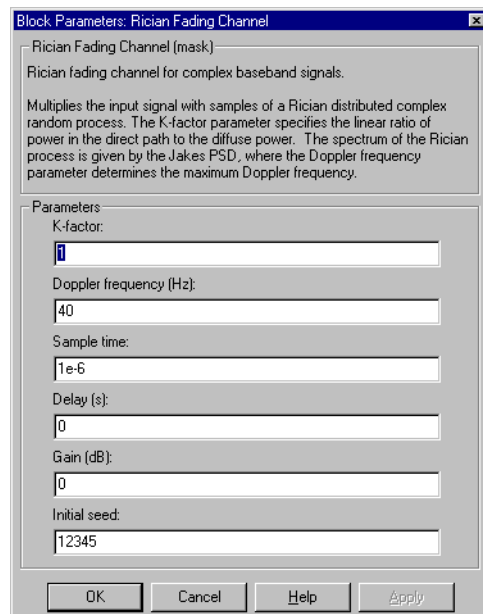
The **Sample time** parameter is the time between successive elements of the input signal. Note that if the input is a frame-based column vector of length n , then the frame period (as Simulink’s Probe block reports, for example) is $n \times \text{Sample time}$.

The **Delay** parameter specifies a time delay in seconds and the **Gain** parameter specifies a gain that applies to the input signal. Both parameters are scalars.

The scalar **Initial seed** parameter seeds the random number generator that the block uses to generate its Rician-distributed complex random process.

Rician Fading Channel

Dialog Box



K-factor

The ratio of power in the direct path to diffuse power. The ratio is expressed linearly, not in decibels.

Doppler frequency (Hz)

A positive scalar that indicates the maximum Doppler shift.

Sample time

The period of each element of the input signal.

Delay (s)

A scalar that specifies the propagation delay.

Gain (dB)

A scalar that specifies the gain.

Initial seed

The scalar seed for the Gaussian noise generator.

See Also Rician Noise Generator, Multipath Rayleigh Fading Channel

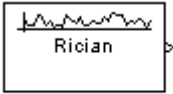
- References**
- [1] Fechtel, Stefan A. "A Novel Approach to Modeling and Efficient Simulation of Frequency-Selective Fading Radio Channels." *IEEE Journal on Selected Areas in Communications*, vol. 11, April 1993. 422-431.
 - [2] Jakes, William C., ed. *Microwave Mobile Communications*. New York: IEEE Press, 1974.
 - [3] Lee, William C. Y. *Mobile Communications Design Fundamentals*, 2nd ed. New York: Wiley, 1993.

Rician Noise Generator

Purpose Generate Rician distributed noise

Library Comm Sources

Description



The Rician Noise Generator block generates Rician distributed noise. The Rician probability density function is given by

$$f(x) = \begin{cases} \frac{x}{\sigma^2} I_0 \frac{mx}{\sigma^2} e^{-\frac{x^2 + m^2}{2\sigma^2}} & x \geq 0 \\ 0 & x < 0 \end{cases}$$

where:

- σ is the standard deviation of the Gaussian distribution that underlies the Rician distribution noise
- $m^2 = m_I^2 + m_Q^2$, where m_I and m_Q are the mean values of two independent Gaussian components
- I_0 is the modified 0th-order Bessel function of the first kind given by

$$I_0(y) = \frac{1}{2\pi} \int_{-\pi}^{\pi} e^{y \cos t} dt$$

Note that m and σ are *not* the mean value and standard deviation for the Rician noise.

You must specify the **Initial seed** for the random number generator. When it is a constant, the resulting noise is repeatable. The vector length of the Initial seed parameter should equal the number of columns in a frame-based output or the number of elements in a sample-based output. The set of numerical parameters above the **Initial seed** parameter in the dialog box can consist of vectors having the same length as the **Initial seed**, or scalars.

Attributes of Output Signal

The output signal can be a frame-based matrix, a sample-based row or column vector, or a sample-based one-dimensional array. These attributes are controlled by the **Frame-based outputs**, **Samples per frame**, and **Interpret**

vector parameters as 1-D parameters. See “Signal Attribute Parameters for Random Sources” on page 2-7 for more details.

The number of elements in the **Initial seed** and **Sigma** parameters becomes the number of columns in a frame-based output or the number of elements in a sample-based vector output. Also, the shape (row or column) of the **Initial seed** and **Sigma** parameters becomes the shape of a sample-based two-dimensional output signal.

Dialog Box

Block Parameters: Rician Noise Generator

Rician Noise Generator (mask)

Generate Rician distributed noise. The output vector size of this block is the same as the vector size of the seed.

Parameters:

Specification method: **K-factor**

Rician K-factor: 2

Sigma: 1

Initial seed: 12345

Sample time: 1

☐ Frame-based outputs

Samples per frame: 1

☐ Interpret vector parameters as 1-D

OK Cancel Help Apply

Specification method

Either **K-factor** or **Quadrature components**.

Rician K-factor

$K = m^2 / (2\sigma^2)$, where m is as in the Rician probability density function. This field appears only if **Specification method** is **K-factor**.

In-phase component (mean), Quadrature component (mean)

The mean values m_I and m_Q , respectively, of the Gaussian components. These fields appear only if **Specification method** is **Quadrature components**.

Rician Noise Generator

Sigma

The variable σ in the Rician probability density function.

Initial seed

The initial seed value for the random number generator.

Sample time

The period of each sample-based vector or each row of a frame-based matrix.

Frame-based outputs

Determines whether the output is frame-based or sample-based. This box is active only if **Interpret vector parameters as 1-D** is unchecked.

Samples per frame

The number of samples in each column of a frame-based output signal. This field is active only if **Frame-based outputs** is checked.

Interpret vector parameters as 1-D

If this box is checked, then the output is a one-dimensional signal. Otherwise, the output is a two-dimensional signal. This box is active only if **Frame-based outputs** is unchecked.

See Also

Rician Fading Channel

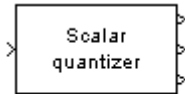
References

[1] Proakis, John G. *Digital Communications*, Third edition. New York: McGraw Hill, 1995.

Purpose Quantize a signal, indicating quantization index, coded signal, and distortion

Library Source Coding

Description



The Sampled Quantizer Encode block encodes an input signal using scalar quantization. The block outputs the quantization levels (or quantization index) of the input signal, the encoded signal, and the mean square distortion.

The input can be either a scalar, a sample-based vector, or a frame-based row vector. This block processes each vector element independently. Each output signal is a vector of the same length as the input signal.

The **Quantization partition** parameter is a length- n real vector whose entries are in strictly ascending order. The first output signal corresponding to an input signal of x is:

- 0 if $x \leq \text{Quantization partition}(1)$
- m if $\text{Quantization partition}(m) < x \leq \text{Quantization partition}(m+1)$
- n if $\text{Quantization partition}(n) < x$

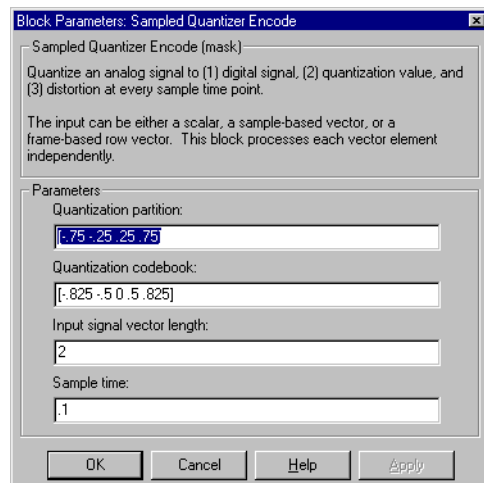
The **Quantization codebook** parameter, whose length exceeds the length of **Quantization partition** by one, prescribes a value for each partition in the quantization. The first element of **Quantization codebook** is the value for the interval between negative infinity and the first element of **Quantization partition**. The second output signal from this block contains the quantization of the input based on the quantization levels and prescribed values.

At a given time, the third output signal measures the mean square distortion between the input and the second output, considering the stream of data up through that time.

You can use the function `loyd` in the Communications Toolbox with a representative sample of your data as training data, to obtain appropriate partition and codebook parameters.

Sampled Quantizer Encode

Dialog Box



Quantization partition

The vector of endpoints of the partition intervals. The elements must be in strictly ascending order.

Quantization codebook

The vector of output values assigned to each partition.

Input signal vector length

The length of the input signal.

Sample time

The output sample time.

Pair Block

Quantizer Decode

See Also

Enabled Quantizer Encode; Lloyd's (Communications Toolbox)

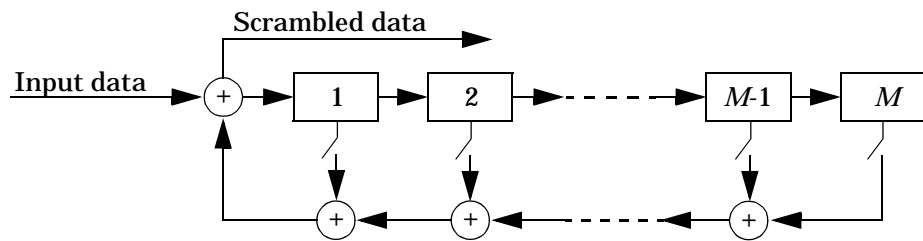
Purpose Scramble the input signal

Library Sequence Operations, in Basic Comm Functions

Description The Scrambler block scrambles the scalar input signal. If the **Calculation base** parameter is N, then the input values must be integers between 0 and N-1.



One purpose of scrambling is to reduce the length of strings of 0s or 1s in a transmitted signal, since a long string of 0s or 1s may cause transmission synchronization problems. Below is a schematic of the scrambler. All adders perform addition modulo N.

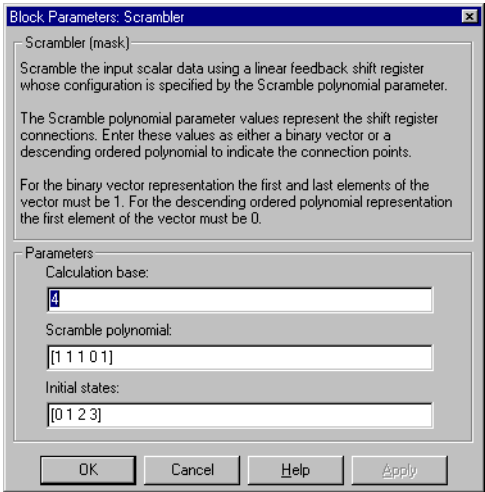


At each time step, the input causes the contents of the registers to shift sequentially. Each switch in the scrambler is on or off as defined by the **Scramble polynomial** parameter. You can specify the polynomial by listing its coefficients in order of ascending powers of z^1 , or by listing the powers of z that appear in the polynomial with a coefficient of 1. For example $p = [1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 1]$ and $p = [0 \ -6 \ -8]$ both represent the polynomial $p(z^1) = 1 + z^6 + z^8$.

The **Initial states** parameter lists the states of the scrambler's registers when the simulation starts. The elements of this vector must be integers between 0 and N-1. The vector length of this parameter must equal the order of the scramble polynomial. (If the **Scramble polynomial** parameter is a vector that lists the coefficients in order, then the order of the scramble polynomial is one less than the vector length.)

Scrambler

Dialog Box



Calculation base

The calculation base N . The input and output of this block are integers in the range $[0, N-1]$.

Scramble polynomial

A polynomial that defines the connections in the scrambler.

Initial states

The states of the scrambler's registers when the simulation starts.

Pair Block

Descrambler

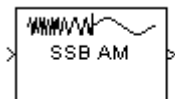
See Also

PN Sequence Generator

Purpose Demodulate SSB-AM-modulated data

Library Analog Baseband Modulation, in Modulation

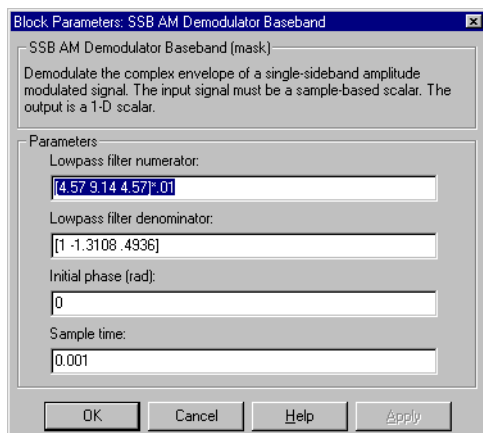
Description



The SSB AM Demodulator Baseband block demodulates a signal that was modulated using single-sideband amplitude modulation. The input is a baseband representation of the modulated signal. The input is complex, while the output is real. The input must be a sample-based scalar signal.

In the course of demodulating, the block uses a filter whose transfer function is described by the **Lowpass filter numerator** and **Lowpass filter denominator** parameters.

Dialog Box



Lowpass filter numerator

The numerator of the lowpass filter transfer function. It is represented as a vector that lists the coefficients in order of descending powers of s .

Lowpass filter denominator

The denominator of the lowpass filter transfer function. It is represented as a vector that lists the coefficients in order of descending powers of s . For an FIR filter, set this parameter to 1.

Initial phase (rad)

The initial phase in the corresponding SSB AM Modulator Baseband block.

SSB AM Demodulator Baseband

Sample time
The sample time of the output signal.

Pair Block SSB AM Modulator Baseband

See Also DSB AM Demodulator Baseband, DSBSC AM Demodulator Baseband

Purpose Demodulate SSB-AM-modulated data

Library Analog Passband Modulation, in Modulation

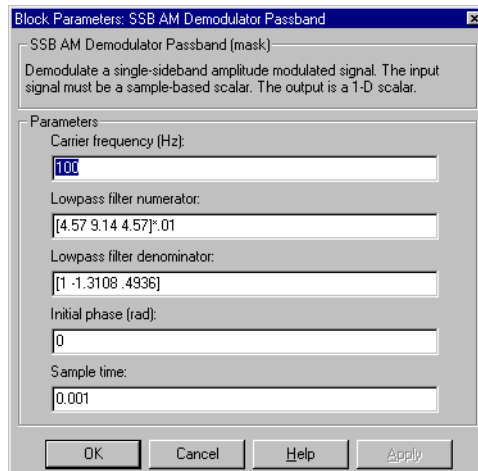
Description



The SSB AM Demodulator Passband block demodulates a signal that was modulated using single-sideband amplitude modulation. The input is a passband representation of the modulated signal. Both the input and output signals are real sample-based scalar signals.

In the course of demodulating, this block uses a filter whose transfer function is described by the **Lowpass filter numerator** and **Lowpass filter denominator** parameters.

Dialog Box



Carrier frequency (Hz)

The carrier frequency in the corresponding SSB AM Modulator Passband block.

Lowpass filter numerator

The numerator of the lowpass filter transfer function. It is represented as a vector that lists the coefficients in order of descending powers of s .

SSB AM Demodulator Passband

Lowpass filter denominator

The denominator of the lowpass filter transfer function. It is represented as a vector that lists the coefficients in order of descending powers of s . For an FIR filter, set this parameter to 1.

Initial phase (rad)

The initial phase of the carrier in radians.

Sample time

The sample time of the output signal.

Pair Block

SSB AM Modulator Passband

See Also

SSB AM Demodulator Baseband, DSB AM Demodulator Passband, DSBSC AM Demodulator Passband

Purpose Modulate using single-sideband amplitude modulation

Library Analog Baseband Modulation, in Modulation

Description



The SSB AM Modulator Baseband block modulates using single-sideband amplitude modulation with a Hilbert transform filter. The output is a baseband representation of the modulated signal. The input signal is real, while the output signal is complex. The input must be a sample-based scalar signal.

SSB AM Modulator Baseband transmits either the lower or upper sideband signal, but not both. To control which sideband it transmits, use the “**upper**” **sideband** or “**lower**” **sideband** parameter.

If the input is $u(t)$ as a function of time t , then the output is

$$(u(t) \mp j\hat{u}(t))e^{j\theta}$$

where θ is the **Initial phase** parameter and $\hat{u}(t)$ is the Hilbert transform of the input $u(t)$. The minus sign indicates the upper sideband and the plus sign indicates the lower sideband.

Hilbert Tranform Filter Parameters

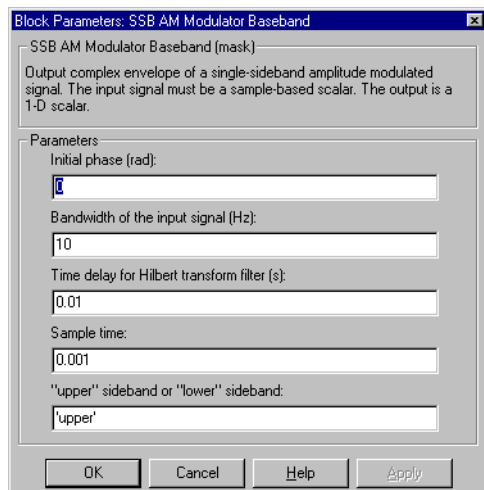
This block uses a Hilbert transform filter, possibly with a compensator. The filter produces a Hilbert transform of its input signal. These mask parameters relate to the Hilbert transform filter:

- The **Time delay for Hilbert transform filter** parameter specifies the delay in the filter design. You should choose a value of the form $(N+1/2)*(\text{Sample time parameter})$ where N is a positive integer.
- The **Bandwidth of the input signal** parameter is the estimated highest frequency component in the input message signal. This parameter is used to design a compensator for the Hilbert transform filter, which would force the message signal amplitude to remain within the assigned range. If this parameter is either 0 or larger than $1/(2*\text{Sample time})$, then the block does not generate a compensator.

SSB AM Modulator Baseband

This block uses the `hilb` function in the Communications Toolbox to design the Hilbert transform filter.

Dialog Box



Initial phase (rad)

The phase offset, θ , of the modulated signal.

Bandwidth of the input signal (Hz)

The highest frequency component of the message signal. To avoid using a compensator in the Hilbert transform filter design, set this to 0.

Time delay for Hilbert transform filter (s)

The time delay in the design of the Hilbert transform filter.

Sample time

The sample time of the Hilbert transform filtering.

“upper” sideband or “lower” sideband

A string that specifies whether to transmit the upper or lower sideband. Choices are 'upper' and 'lower'.

Pair Block

SSB AM Demodulator Baseband

See Also

DSB AM Modulator Baseband, DSBSC AM Modulator Baseband

References

[1] Peebles, Peyton Z, Jr. *Communication System Principles*. Reading, Mass.: Addison-Wesley, 1976.

SSB AM Modulator Passband

Purpose Modulate using single-sideband amplitude modulation

Library Analog Passband Modulation, in Modulation

Description



The SSB AM Modulator Passband block modulates using single-sideband amplitude modulation with a Hilbert transform filter. The output is a passband representation of the modulated signal. Both the input and output signals are real sample-based scalar signals.

SSB AM Modulator Passband transmits either the lower or upper sideband signal, but not both. To control which sideband it transmits, use the “**upper**” **sideband** or “**lower**” **sideband** parameter.

If the input is $u(t)$ as a function of time t , then the output is

$$u(t) \cos(f_c t + \theta) \pm \hat{u}(t) \sin(f_c t + \theta)$$

where:

- f_c is the **Carrier frequency** parameter.
- θ is the **Initial phase** parameter.
- $\hat{u}(t)$ is the Hilbert transform of the input $u(t)$.
- The plus sign indicates the upper sideband and the minus sign indicates the lower sideband.

Hilbert Transform Filter Parameters

This block uses a Hilbert transform filter, possibly with a compensator. These mask parameters relate to the Hilbert transform filter:

- The **Time delay for Hilbert transform filter** parameter specifies the delay in the filter design. You should choose a value of the form $(N+1/2) \cdot (\text{Sample time parameter})$ where N is a positive integer.
- The **Bandwidth of the input signal** parameter is the estimated highest frequency component in the input message signal.

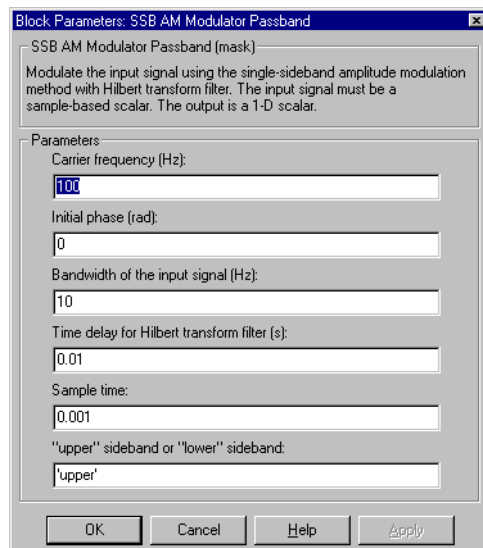
This parameter is used to design a compensator for the Hilbert transform filter, which would force the message signal amplitude to remain within the

assigned range. If this parameter is either 0 or larger than $1/(2 \cdot \text{Sample time})$, then the block does not generate a compensator.

This block uses the `hilbert` function in the Communications Toolbox to design the Hilbert transform filter.

Typically, an appropriate **Carrier frequency** value is much higher than the highest frequency of the input signal. To avoid having to use a high carrier frequency and consequently a high sampling rate, you can use baseband simulation (SSB AM Modulator Baseband block) instead of passband simulation.

Dialog Box



Carrier frequency (Hz)

The frequency of the carrier.

Initial phase (rad)

The phase offset, θ , of the modulated signal.

Bandwidth of the input signal (Hz)

The highest frequency component of the message signal. To avoid using a compensator in the Hilbert transform filter design, set this to 0.

SSB AM Modulator Passband

Time delay for Hilbert transform filter (s)

The time delay in the design of the Hilbert transform filter.

Sample time

The sample time of the Hilbert transform filtering.

“upper” sideband or “lower sideband”

A string that specifies whether to transmit the upper or lower sideband. Choices are 'upper' and 'lower'.

Pair Block

SSB AM Demodulator Passband

See Also

SSB AM Modulator Baseband, DSB AM Modulator Passband, DSBSC AM Modulator Passband; `hilbtr` (Communications Toolbox)

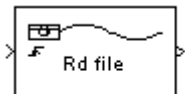
References

[1] Peebles, Peyton Z, Jr. *Communication System Principles*. Reading, Mass.: Addison-Wesley, 1976.

Purpose Read from a file, refreshing the output at rising edges of an input signal

Library Comm Sources

Description The Triggered Read From File block reads a new record from a file *only* at the rising edge of the input trigger signal. The output is a sample-based signal.



Note The triggered behavior of this block is one difference between this block and Simulink's From File block. However, the From File block is useful for reading platform-independent MAT-files.

The file can be an ASCII text file, a file containing integer or floating point numbers, or a binary file (in the format of the C `fwrite` function). The file must be either in the current working directory or the MATLAB path.

When a rising edge of the input trigger signal is detected, this block reads from the file a record whose length is specified in the parameter **Output vector length**. The first reading always occurs at the first rising edge. After that, if the **Decimation** parameter is a positive integer k , then the block reads at every k th rising edge. If **Decimation** is 1, then the block reads at every rising edge.

When the block reaches the end-of-file marker, it either:

- Rereads from the beginning of the file, if the **Cyclic repeat** box is checked, or
- Outputs zeros, if the **Cyclic repeat** box is not checked

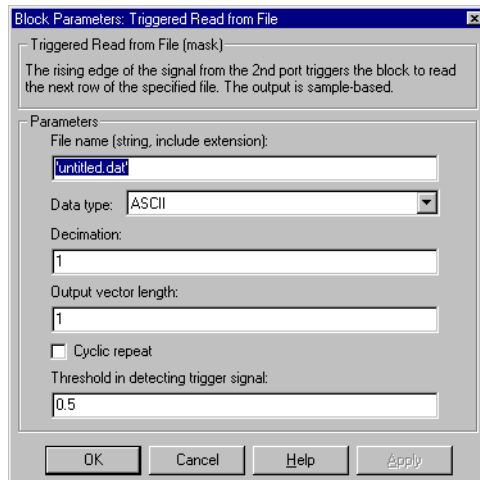
If the **Data type** parameter is **ASCII**, then the output is an integer. The mapping between decimal integers and ASCII characters is shown below.

Integer	ASCII	Integer	ASCII	Integer	ASCII	Integer	ASCII
0	NUL	32	SP	64	@	96	\
1	SOH	33	!	65	A	97	a
2	STX	34	"	66	B	98	b
3	ETX	35	#	67	C	99	c
4	EOT	36	\$	68	D	100	d

Triggered Read From File

Integer	ASCII	Integer	ASCII	Integer	ASCII	Integer	ASCII
5	ENQ	37	%	69	E	101	e
6	ACK	38	&	70	F	102	f
7	BEL	39	'	71	G	103	g
8	BS	40	(72	H	104	h
9	HT	41)	73	I	105	i
10	LF	42	*	74	J	106	j
11	VT	43	+	75	K	107	k
12	FF	44	,	76	L	108	l
13	CR	45	-	77	M	109	m
14	SO	46	.	78	N	110	n
15	SI	47	/	79	O	111	o
16	DLE	48	0	80	P	112	p
17	DC1	49	1	81	Q	113	q
18	DC2	50	2	82	R	114	r
19	DC3	51	3	83	S	115	s
20	DC4	52	4	84	T	116	t
21	NAK	53	5	85	U	117	u
22	SYN	54	6	86	V	118	v
23	ETB	55	7	87	W	119	w
24	CAN	56	8	88	X	120	x
25	EM	57	9	89	Y	121	y
26	SUB	58	:	90	Z	122	z
27	ESC	59	;	91	[123	{
28	FS	60	<	92	\	124	
29	GS	61	=	93]	125	}
30	RS	62	>	94	^	126	~
31	US	63	?	95	_	127	DEL

Dialog Box



File name

The filename, including its extension, as a string.

Data type

The data type. Choices are **ASCII**, **binary**, **float**, and **integer**.

Decimation

A decimation factor. If it is 1, then the block reads at every rising edge.

Output vector length

The vector length of the block's output.

Cyclic repeat

Specifies whether to cycle continuously through the contents of **File name** or to output only zeros after reaching the end-of-file marker.

Threshold in detecting trigger signal

The threshold for the rising edge of the trigger signal.

Pair Block

Triggered Write to File

See Also

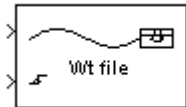
To File (Simulink)

Triggered Write to File

Purpose Write to a file at each rising edge of an input signal

Library Comm Sinks

Description



The Triggered Write to File block creates a file containing selected data from the input signal. Unlike Simulink's To File block, the Triggered Write to File block writes new data to the file *only* at the rising edge of the input trigger signal. However, the To File block is useful for creating platform-independent MAT-files.

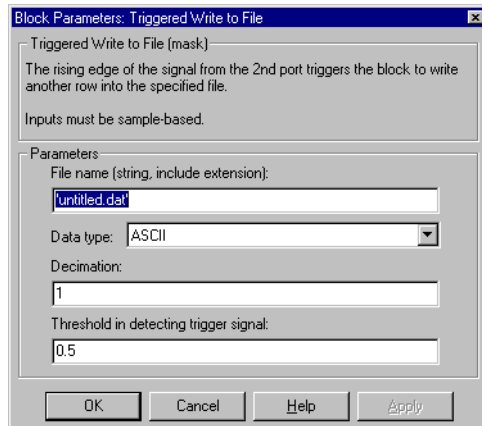
The file can be an ASCII text file, a file containing integer or floating-point numbers, or a binary file (in the format of the C `fwrite` function). You specify the file type using the **Data type** parameter. If **Data type** is **ASCII** then this block converts the data into ASCII characters before writing, using the mapping shown on the reference page for the Triggered Read From File block. For example, an input of 65 would cause the the block to write the character "A" to the file. Other file types receive the data directly.

Caution If the destination file already exists, then this block *overwrites* it.

The first input signal contains the data to write. This input must be sample-based. The second input signal is a sample-based scalar trigger signal that controls the timing of writing. When a rising edge of the input trigger signal is detected, this block writes the elements of the data signal to the file. The file does not contain information about the dimension or orientation of the data input, however.

The first write always occurs at the first rising edge. After that, the **Decimation** parameter determines how many triggers the block receives between successive file writes. Setting this parameter to 1 causes the block to write at every rising edge.

Dialog Box



File name

The filename, including its extension, as a string.

Data type

The data type. Choices are **ASCII**, **binary**, **float**, and **integer**.

Decimation

A decimation factor. If it is 1, then the block writes at every rising edge.

Threshold in detecting trigger signal

The threshold for the rising edge of the trigger signal.

Pair Block

Triggered Read From File

See Also

To File (Simulink)

Uniform Noise Generator

Purpose Generate uniformly distributed noise between the upper and lower bounds

Library Comm Sources

Description



The Uniform Noise Generator block generates uniformly distributed noise. The output data of this block is uniformly distributed between the specified lower and upper bounds. The upper bound must be greater than or equal to the lower bound.

You must specify the **Initial seed** in the simulation. When it is a constant, the resulting noise is repeatable.

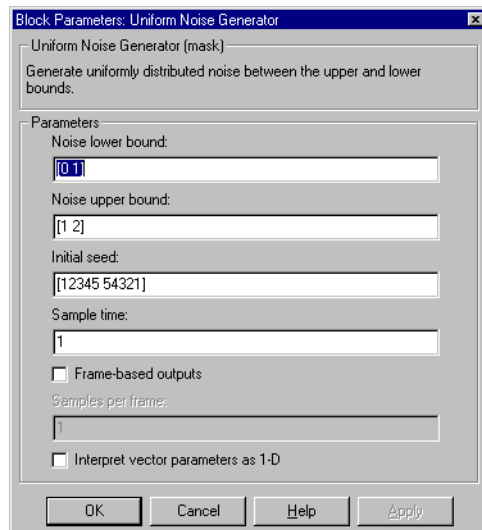
If all the elements of the output vector are to be independent and identically distributed (i.i.d.), then you can use a scalar for the **Noise lower bound** and **Noise upper bound** parameters. Alternatively, you can specify the range for each element of the output vector individually, by using vectors for the **Noise lower bound** and **Noise upper bound** parameters. If the bounds are vectors, then their length must equal the length of the **Initial seed** parameter.

Attributes of Output Signal

The output signal can be a frame-based matrix, a sample-based row or column vector, or a sample-based one-dimensional array. These attributes are controlled by the **Frame-based outputs**, **Samples per frame**, and **Interpret vector parameters as 1-D** parameters. See “Signal Attribute Parameters for Random Sources” on page 2-7 for more details.

The number of elements in the **Initial seed** parameter becomes the number of columns in a frame-based output or the number of elements in a sample-based vector output. Also, the shape (row or column) of the **Initial seed** parameter becomes the shape of a sample-based two-dimensional output signal.

Dialog Box



Noise lower bound, Noise upper bound

The lower and upper bounds of the interval over which noise is uniformly distributed.

Initial seed

The initial seed value for the random number generator.

Sample time

The period of each sample-based vector or each row of a frame-based matrix.

Frame-based outputs

Determines whether the output is frame-based or sample-based. This box is active only if **Interpret vector parameters as 1-D** is unchecked.

Samples per frame

The number of samples in each column of a frame-based output signal. This field is active only if **Frame-based outputs** is checked.

Uniform Noise Generator

Interpret vector parameters as 1-D

If this box is checked, then the output is a one-dimensional signal.

Otherwise, the output is a two-dimensional signal. This box is active only if **Frame-based outputs** is unchecked.

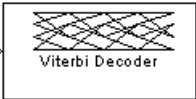
See Also

Random Source (DSP Blockset); rand (built-in MATLAB function)

Purpose Decode convolutionally encoded data using the Viterbi algorithm

Library Convolutional sublibrary of Channel Coding

Description The Viterbi Decoder block decodes input symbols to produce binary output symbols. This block can process several symbols at a time for faster performance.



Input and Output Sizes

If the convolutional code uses an alphabet of 2^n possible symbols, then this block's input vector length is $L \cdot n$ for some positive integer L . Similarly, if the decoded data uses an alphabet of 2^k possible output symbols, then this block's output vector length is $L \cdot k$. The integer L is the number of frames that the block processes in each step.

The input can be either a sample-based vector with $L = 1$, or a frame-based column vector with any positive integer for L .

Input Values and Decision Types

The entries of the input vector are either bipolar, binary, or integer data, depending on the **Decision type** parameter.

Decision type Parameter	Possible Entries in Decoder Input	Interpretation of Values
Unquantized	Real numbers	+1: logical zero
		-1: logical one

Viterbi Decoder

Decision type Parameter	Possible Entries in Decoder Input	Interpretation of Values
Hard Decision	0, 1	0: logical zero 1: logical one
Soft Decision	Integers between 0 and 2^b-1 , where b is the Number of soft decision bits parameter	0: most confident decision for logical zero 2^b-1 : most confident decision for logical one Other values represent less confident decisions

To illustrate the soft decision situation more explicitly, the table below lists interpretations of values for 3-bit soft decisions.

Input Value	Interpretation
0	Most confident zero
1	Second most confident zero
2	Third most confident zero
3	Least confident zero
4	Least confident one
5	Third most confident one
6	Second most confident one
7	Most confident one

Operation Modes for Frame-Based Inputs

If the input signal is frame-based, then the block has three possible methods for transitioning between successive frames. The **Operation mode** parameter controls which method the block uses:

- In **Continuous** mode, the block saves its internal state metric at the end of each frame, for use with the next frame. Each traceback path is treated independently.
- In **Truncated** mode, the block treats each frame independently. The traceback path starts at the state with the best metric and always ends in the all-zeros state. This mode is appropriate when the corresponding Convolutional Encoder block has its **Reset** parameter set to **On each frame**.
- In **Terminated** mode, the block treats each frame independently, and the traceback path always starts and ends in the all-zeros state. This mode is appropriate when the uncoded message signal (that is, the input to the corresponding Convolutional Encoder block) has enough zeros at the end of each frame to fill all memory registers of the encoder. If the encoder has k input streams and constraint length vector constr (using the polynomial description), then “enough” means $k * \max(\text{constr} - 1)$.

In the special case when the frame-based input signal contains only one symbol, the **Continuous** mode is most appropriate.

Traceback Depth and Decoding Delay

The **Traceback depth** parameter, D , influences the decoding delay. The decoding delay is the number of zero symbols that precede the first decoded symbol in the output.

- If the input signal is sample-based, then the decoding delay consists of D zero symbols
- If the input signal is frame-based and the **Operation mode** parameter is set to **Continuous**, then the decoding delay consists of D zero symbols
- If the **Operation mode** parameter is set to **Truncated** or **Terminated**, then there is no output delay and the **Traceback depth** parameter must be less than or equal to the number of symbols in each frame.

If the code rate is $1/2$, then a typical **Traceback depth** value is about five times the constraint length of the code.

Reset Port

The reset port is usable only when the **Operation mode** parameter is set to **Continuous**. Checking the **Reset input** check box causes the block to have an

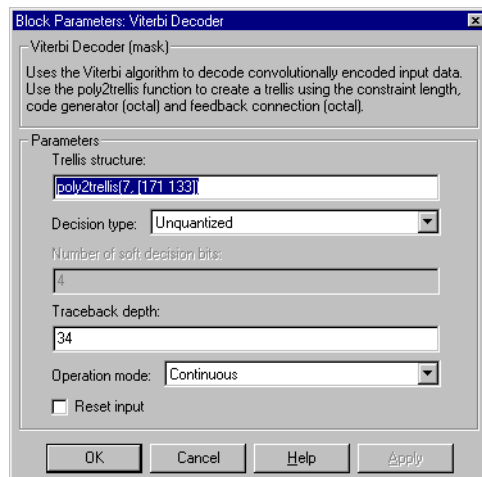
Viterbi Decoder

additional input port, labeled **Rst**. When the **Rst** input is nonzero, the decoder returns to its initial state by configuring its internal memory as follows:

- Sets the all-zeros state metric to zero
- Sets all other state metrics to the maximum value
- Sets the traceback memory to zero

Using a reset port on this block is analogous to setting the **Reset** parameter in the Convolutional Encoder block to **On nonzero Rst input**.

Dialog Box



Trellis structure

MATLAB structure that contains the trellis description of the convolutional encoder. Use the same value here and in the corresponding Convolutional Encoder block.

Decision type

Unquantized, **Hard Decision**, or **Soft Decision**.

Number of soft decision bits

The number of soft decision bits used to represent each input. This field is active only when **Decision type** is set to **Soft Decision**.

Traceback depth

The number of trellis branches used to construct each traceback path.

Operation mode

Method for transitioning between successive input frames. For frame-based input, the choices are **Continuous**, **Terminated**, and **Truncated**. Sample-based input must use the **Continuous** mode.

Reset input

When you check this box, the decoder has a second input port labeled Rst. Providing a nonzero input value to this port causes the internal memory to be set to its initial state prior to processing the input data.

See Also Convolutional Encoder, APP Decoder

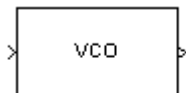
- References**
- [1] Clark, George C. Jr. and J. Bibb Cain. *Error-Correction Coding for Digital Communications*. New York: Plenum Press, 1981.
 - [2] Gitlin, Richard D., Jeremiah F. Hayes, and Stephen B. Weinstein. *Data Communications Principles*. New York: Plenum, 1992.
 - [3] Heller, Jerrold A. and Irwin Mark Jacobs. "Viterbi Decoding for Satellite and Space Communication." *IEEE Transactions on Communication Technology*, vol. COM-19, October 1971. 835-848.

Voltage-Controlled Oscillator

Purpose Implement a voltage-controlled oscillator

Library Comm Sources

Description



The Voltage-Controlled Oscillator (VCO) block generates a signal whose frequency shift from the **Oscillation frequency** parameter is proportional to the input signal. The input signal is interpreted as a voltage. If the input signal is $u(t)$, then the output signal is

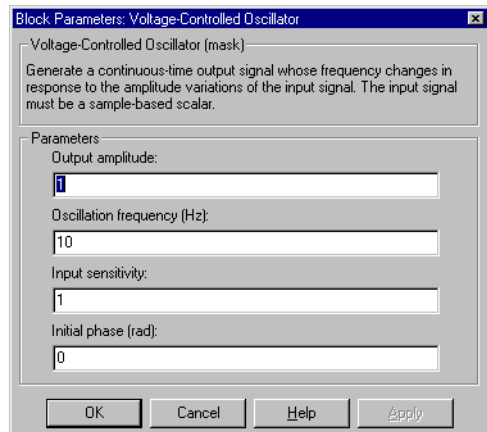
$$y(t) = A_c \cos(2\pi f_c t + 2\pi k_c \int_0^t u(\tau) d\tau + \varphi)$$

where A_c is the **Output amplitude** parameter, f_c is the **Oscillation frequency** parameter, k_c is the **Input sensitivity** parameter, and φ is the **Initial phase** parameter.

This block uses a continuous-time integrator to interpret the equation above.

The input and output signals are both sample-based scalars.

Dialog Box



Output amplitude

The amplitude of the output.

Oscillation frequency (Hz)

The frequency of the oscillator output when the input signal is zero.

Input sensitivity

This value scales the input voltage and, consequently, the shift from the **Oscillation frequency** value. The units of **Input sensitivity** are Hertz per volt.

Initial phase (rad)

The initial phase of the oscillator in radians.

See Also

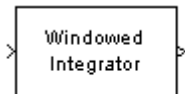
Discrete-Time VCO

Windowed Integrator

Purpose Integrate over a time window of fixed length

Library Integrators, in Basic Comm Functions

Description

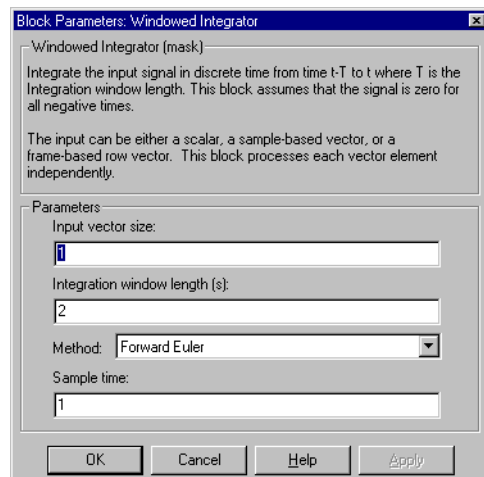


The Windowed Integrator block integrates the input signal in discrete time, over a sliding time window of fixed length. If the **Integration window length** parameter is T , then the output at time t is the result of integrating the input signal from $t-T$ to t . The block assumes that the input signal is zero for all negative t .

You can choose one of three integration methods: **Forward Euler**, **Backward Euler**, and **Trapezoidal**.

The input can be either a scalar, a sample-based vector, or a frame-based row vector. The block processes each vector element independently. If the input signal is a vector, then the output is a vector of the same length. This length appears as the **Input vector size** parameter.

Dialog Box



Input vector size

The length of the input vector.

Integration window length (s)

The length of the interval of integration, in seconds.

Method

The integration method. Choices are **Forward Euler**, **Backward Euler**, and **Trapezoidal**.

Sample time

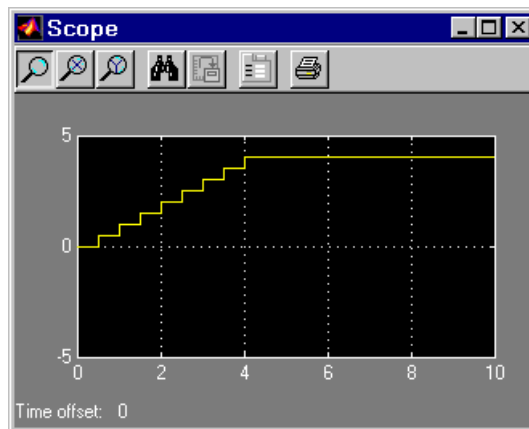
The integration sample time. This must not exceed the **Integration window length** parameter.

Examples

Integrate a scalar constant signal whose value is 1 using these parameters:

- **Input vector size** = 5
- **Integration window length** = 4
- **Method** = **Forward Euler**
- **Sample time** = . 5

You can use a Simulink Constant block for the input signal. The Simulink Scope block shows the output below.



Notice that the output from time 0 to time 4 is a discrete approximation of a ramp. During this period, the interval of integration increases with time. Also notice that the output after time 4 is a constant value of 4. This is the result of integrating the value 1 over the full integration window length of 4 seconds.

See Also

Discrete Modulo Integrator, Integrate and Dump, Discrete-Time Integrator (Simulink)

Windowed Integrator

A

- A-law companders 2-24
- A-Law Compressor block 4-47
- A-Law Expander block 4-49
- Algebraic Deinterleaver block 4-51
- Algebraic Interleaver block 4-53
- amplitude modulation (AM)
 - example model 2-59
- analog modulation libraries 2-53
 - reference for 4-24
- analog-to-digital conversion 2-16
- APP Decoder block 4-56
- AWGN Channel block 4-60

B

- baseband
 - simulation 2-54
- Baseband PLL block 4-64
- Basic Comm Functions library 4-37
- BCH Decoder block 4-66
- BCH Encoder block 4-68
- Bernoulli Random Binary Generator block 4-70
- binary
 - codes 2-28
 - numbers, order of digits and 2-31
 - vector format for messages and codewords 2-29
 - in Reed-Solomon codes 2-30
- Binary Cyclic Decoder block 4-73
- Binary Cyclic Encoder block 4-75
- Binary Linear Decoder block 4-79
- Binary Linear Encoder block 4-81
- Binary Symmetric Channel block 4-84
- Binary Vector Noise Generator block 4-86
- Binary-Input RS Encoder block 4-77
- Binary-Output RS Decoder block 4-82

- Bit to Integer Converter block 4-89
- block coding
 - features of blockset 2-28
 - methods supported in blockset 2-28
 - techniques for 2-28
 - terminology and notation 2-29
- block coding library 2-27
 - reference for 4-9
- block interleaving library 2-46
 - reference for 4-13
- BPSK Demodulator Baseband block 4-90
- BPSK Modulator Baseband block 4-92

C

- Channel Coding library 2-27
 - reference for 4-9
- Channels library 2-84
 - reference for 4-34
- Charge Pump PLL block 4-94
- code generator matrices
 - representing 2-36
- codebooks
 - representing 2-17
- codewords
 - definition 2-29
 - representing 2-29
- coding, source
 - features of the blockset 2-16
- column vector signals xviii
- Comm Sinks library 2-12
 - reference for 4-5
- Comm Sources library 2-6
 - reference for 4-3
- companders 2-24
 - example 2-24

Complex Phase Difference block 4-97
Complex Phase Shift block 4-98
compression
 of data 2-16
compressors 2-24
 example 2-24
Continuous-Time Eye and Scatter Diagrams
 block 4-99
converting
 analog to digital 2-16
convolutional coding
 delays 2-42
convolutional coding library 2-40
 reference for 4-11
Convolutional Deinterleaver block 4-103
Convolutional Encoder block 4-105
Convolutional Interleaver block 4-107
convolutional interleaving library 2-47
 reference for 4-15
CPFSK Demodulator Baseband block 4-109
CPFSK Demodulator Passband block 4-112
CPFSK Modulator Baseband block 4-115
CPFSK Modulator Passband block 4-118
CPM Demodulator Baseband block 4-121
CPM Demodulator Passband block 4-125
CPM Modulator Baseband block 4-130
CPM Modulator Passband block 4-134

D

data compression 2-16
Data Mapper block 4-139
DBPSK Demodulator Baseband block 4-142
DBPSK Modulator Baseband block 4-144
decision timing
 and eye diagrams 2-13
 and scatter diagrams 2-13

Deinterlacer block 4-146
delays
 from analog demodulator's filter 2-62
 in convolutional coding 2-42
 in digital modulation 2-69
 in example model 1-25
 in interleaving 2-49
 in serial-signal channel coding 2-30
delta modulation 2-21
 example 2-22
 See also differential pulse code modulation
demodulation
 definition of 2-53
Derepeat block 4-147
Descrambler block 4-150
diagrams
 eye 2-13
 example 2-14
 scatter 2-13
Differential Decoder block 4-152
Differential Encoder block 4-153
differential pulse code modulation (DPCM) 2-21
 example 2-22
 parameters, representing 2-17
digital modulation libraries 2-64
 reference for 4-18
Discrete Modulo Integrator block 4-154
Discrete-Time Eye and Scatter Diagrams block
 4-156
Discrete-Time VCO block 4-159
DPCM Decoder block 4-161
DPCM Encoder block 4-163
DQPSK Demodulator Baseband block 4-165
DQPSK Modulator Baseband block 4-167
DSB AM Demodulator Baseband block 4-171
DSB AM Demodulator Passband block 4-173
DSB AM Modulator Baseband block 4-175

DSB AM Modulator Passband block 4-176
 DSBSC AM Demodulator Baseband block 4-178
 DSBSC AM Demodulator Passband block 4-180
 DSBSC AM Modulator Baseband block 4-182
 DSBSC AM Modulator Passband block 4-183

E

Enabled Quantizer Encode block 4-185
 error
 predictive 2-21
 Error Rate Calculation block 4-187
 error-control coding
 base 2 only 2-28
 error-correction capability
 of Hamming codes 2-36
 of Reed-Solomon codes 2-38
 examples
 analog modulation step sizes 2-57
 analog modulation timing 2-55
 companders 2-24
 continuous phase modulation 2-78
 convolutional coding 2-42
 convolutional interleaving 2-50
 delays from filtering 2-62
 delays in digital demodulation 2-71
 differential pulse code modulation 2-22
 eye and scatter diagrams 2-14
 fading channels 2-86
 filter cutoffs 2-59
 Hamming coding 2-32
 passband digital modulation 2-81
 quantization 2-18
 quantized sine wave 2-19
 Reed-Solomon coding 2-33
 scatter diagrams 2-76
 signal constellations 2-75

expanders 2-24
 example 2-24
 eye diagrams 2-13
 example 2-14

F

features
 analog modulation 2-53
 block coding 2-28
 channel 2-84
 digital modulation 2-65
 interleaving 2-46
 sink 2-12
 source 2-6
 source coding 2-16
 synchronization 2-90
 filters
 use after demodulating
 choosing cutoff frequency 2-59
 resulting time lag 2-62
 FM Demodulator Baseband block 4-194
 FM Demodulator Passband block 4-196
 FM Modulator Baseband block 4-198
 FM Modulator Passband block 4-200
 frame attribute xviii
 frame-based signals, definition of xviii
 full matrix signal, definition of xviii

G

Gaussian Noise Generator block 4-202
 General Block Deinterleaver block 4-205
 General Block Interleaver block 4-207
 General Multiplexed Deinterleaver block 4-208
 General Multiplexed Interleaver block 4-210

General QAM Demodulator Baseband block
4-212
General QAM Demodulator Passband block
4-214
General QAM Modulator Baseband block 4-217
General QAM Modulator Passband block 4-219
generator matrices
 representing 2-36
GMSK Demodulator Baseband block 4-222
GMSK Demodulator Passband block 4-225
GMSK Modulator Baseband block 4-228
GMSK Modulator Passband block 4-231

H

Hamming coding
 for single-error-correction 2-36
Hamming Decoder block 4-234
Hamming Encoder block 4-236
Helical Deinterleaver block 4-238
Helical Interleaver block 4-241

I

Insert Zero block 4-244
integer format for messages and codewords 2-31
Integer to Bit Converter block 4-250
Integer-Input RS Encoder block 4-246
Integer-Output RS Decoder block 4-248
Integrate and Dump block 4-251
Integrators library 4-37
Interlacer block 4-253
interleaving delays 2-49
Interleaving library 2-46, 4-13

L

linear predictors
 representing 2-22
Linearized Baseband PLL block 4-254

M

Matrix Deinterleaver block 4-256
Matrix Helical Scan Deinterleaver block 4-257
Matrix Helical Scan Interleaver block 4-259
Matrix Interleaver block 4-262
M-DPSK Demodulator Baseband block 4-264
M-DPSK Demodulator Passband block 4-267
M-DPSK Modulator Baseband block 4-270
M-DPSK Modulator Passband block 4-274
messages
 definition 2-29
 representing 2-29
M-FSK Demodulator Baseband block 4-277
M-FSK Demodulator Passband block 4-280
M-FSK Modulator Baseband block 4-283
M-FSK Modulator Passband block 4-286
 μ -law companders 2-24
 example 2-24
modulation
 analog 2-53
 methods supported in blockset 2-53
 definition of 2-53
 delta 2-21
 example 2-22
 See also differential pulse code modulation
 differential pulse code. *See* differential pulse
 code modulation
 digital 2-64
 methods supported in blockset 2-65
Modulation library 2-53
 reference for 4-18

Modulo Integrator block 4-289
 M-PAM Demodulator Baseband block 4-290
 M-PAM Demodulator Passband block 4-293
 M-PAM Modulator Baseband block 4-297
 M-PAM Modulator Passband block 4-301
 M-PSK Demodulator Baseband block 4-305
 M-PSK Demodulator Passband block 4-308
 M-PSK Modulator Baseband block 4-311
 M-PSK Modulator Passband block 4-316
 MSK Demodulator Baseband block 4-319
 MSK Demodulator Passband block 4-321
 MSK Modulator Baseband block 4-324
 MSK Modulator Passband block 4-326
 Mu-Law Compressor block 4-329
 Mu-Law Expander block 4-330
 Multipath Rayleigh Fading Channel block 4-331

N

nonbinary codes 2-28
 Reed-Solomon 2-38

O

one-dimensional arrays, definition of xviii
 OQPSK Demodulator Baseband block 4-334
 OQPSK Demodulator passband block 4-336
 OQPSK Modulator Baseband block 4-339
 OQPSK Modulator Passband block 4-342
 order of digits in binary numbers 2-31
 orientations, of vector signals xviii

P

$\pi/4$ DQPSK modulation 2-75
 partitions
 representing 2-17

passband
 simulation 2-54
 phase modulation (PM)
 example model 2-62
 Phase-Locked Loop block 4-345
 $\pi/4$ DQPSK modulation 2-75
 PLLs
 features of the blockset 2-90
 PM Demodulator Baseband block 4-348
 PM Demodulator Passband block 4-350
 PM Modulator Baseband block 4-352
 PM Modulator Passband block 4-353
 PN Sequence Generator block 4-355
 Poisson Int Generator block 4-358
 predictive
 error 2-21
 predictive quantization 2-21
 example 2-22
 features of the blockset 2-16
 parameters, representing 2-17
 predictors 2-21
 representing 2-22
 Puncture block 4-360

Q

QPSK Demodulator Baseband block 4-362
 QPSK Modulator Baseband block 4-364
 quantization
 coding 2-20
 example 2-18
 features of the blockset 2-16
 parameters, representing 2-17
 predictive 2-21
 example 2-22
 vector 2-16
 Quantizer Decode block 4-367

R

random

signals 2-6

Random Deinterleaver block 4-368

Random Interleaver block 4-372

Random-Integer Generator block 4-369

Rayleigh Noise Generator block 4-373

Rectangular QAM Demodulator Baseband block
4-376Rectangular QAM Demodulator Passband block
4-379Rectangular QAM Modulator Baseband block
4-383Rectangular QAM Modulator Passband block
4-387

references

error-correction coding 2-39

representing

codebooks 2-17

codewords 2-29

generator matrices 2-36

messages 2-29

partitions 2-17

predictors 2-22

quantization parameters 2-17

truth tables 2-36

Rician Fading Channel block 4-391

Rician Noise Generator block 4-394

row vector signals xviii

S

sample times, in sources 2-7

sample-based signals, definition of xviii

Sampled Quantizer Encode block 4-397

scalar quantization

coding 2-20

example 2-18

features of the blockset 2-16

parameters, representing 2-17

scalar signals

definition of xviii

scatter diagrams 2-13

Scrambler block 4-399

Sequence Operations library 4-38

signal formatting

features of the blockset 2-16

Signal Processing Toolbox

for filter design 2-59

sinks library 2-12

reference for 4-5

sizes, of matrix signals xviii

source coding

features of the blockset 2-16

Source Coding library 2-16

reference for 4-7

sources library 2-6

reference for 4-3

SSB AM Demodulator Baseband block 4-401

SSB AM Demodulator Passband block 4-403

SSB AM Modulator Baseband block 4-405

SSB AM Modulator Passband block 4-408

synchronization

features of the blockset 2-90

Synchronization library 2-90

reference for 4-36

T

timing, decision

and eye diagrams 2-13

and scatter diagrams 2-13

truth tables

representing 2-36

U

Uniform Noise Generator block 4-416

Utility Functions library 4-41

V

vector quantization 2-16

vector signals, definition of xviii

Viterbi Decoder block 4-419

Voltage-Controlled Oscillator block 4-424

W

Windowed Integrator block 4-426